UNIVERSITY OF OSLO
Department of Informatics

Authenticating HTTPS servers through the use of DNS in an Offline Personal Authentication Device (OffPAD)

Joakim Hovlandsvåg
<joakimsh@ifi.uio.no>

August 1, 2013

Abstract

This Master's project investigates how to strengthen the server authentication for end users using an *Offline Personal Authentication Device (OffPAD)*, especially when combined with DNSSEC. The Master's project is a part of the *Local User-Centric Identity Management (LUCIDMAN) project*[1], which is a Franco-Norwegian research project seeking to strengthen the security usability for end users. The research project's main focus is an *OffPAD*, which is a physical device that should be mainly offline and should help the user in the authentication and transaction process when using services online on a computer or when in a physical location like a shop. The device contains functionality to help the end user in securing its activity and transactions, protecting its credentials and avoid phishing and network attacks.

The goal of this thesis is to find a solution to authentication of online servers when the authentication process happens in an offline device. The information required for the authentication process must be transferred through an untrusted online device, like the user's computer, that might be infected with Trojans, or an online device in the shop, that might have been tampered with.

The proposal is motivated by the increasing lack of trust in the Public-Key Infrastructure using X.509 that is used for HTTPS today [21], and the increased trust and popularity in using DNSSEC for authenticated information. By using DNSSEC for storing authenticated information, the OffPAD could make use of the newly standardized *TLSA* specification, which defines how to store certificates in DNS and how to use them in HTTPS and for other protocols. The solution should still be able to use the current X.509 PKI for servers that is not set up with DNSSEC or TLSA.

The proposed solutions will in parts be useful for online entities too, as DNSSEC have been blocked in some networks, and might be slow to process for clients with a requirement of short response times, like web browsers.

For a service to make use of this proposal, it must be hosted in a DNS zone that supports DNSSEC, and its administrators must be able to register its public key data into DNS. The server software itself does not need to be modified.

[1] www.lucidman.org

Contents

Preface

Takk for tolmodet eg er gitt i prosessen med å fullføre masteroppgåva mi. Det har tatt lang tid og mange kveldar med skriving, spesielt når eg har hatt privilegiet av ein krevande og spennande fulltidsjobb ved sidan av.

Særskild nemnde:

- Audun Jøsang, for fagleg inspirasjon, veileding og interesse.
- kentav, hennikl, jazz, mocca, amveha og andre kollegaer, for motivasjon.
- Wilma, for glede og tankepausar.
- Mari, for alt.

Part I

Introduction

Chapter 1

Motivation

When doing online transactions, there is always a risk that they get compromised. Important transactions, like online banking, are especially attractive for attackers. There are many ways to get attacked, and the criminals are continuously using creativity to find new ways. Your computer could be infected by a Trojan, you could be a victim of a phishing attack, you could be tricked into going to the wrong website, or an attacker could even tamper with your network data to fool both you and your computer. When being attacked, you might not even know it. The false web site you are sending your credentials to might look identical to the real one, both to you and the computer.

One attempt to slow down this development, is for the users to have an *Offline authentication device (OffPAD)*[1], which is doing the authentication for the user by storing the credentials for each service. The device is then offline to lower the risk of being attacked, and communicates only through your computer or other transaction devices like payment devices in shops. When credentials are needed, your computer or the payment device must ask the OffPAD for the credentials. The OffPAD authenticates the service and asks the end user for confirmation to hand over the credentials.

Such a device would protect the end user from various attacks, in addition to making it easier for the user, as different passwords are not needed to be remembered for each service. Trojans would have a harder time of getting into the offline device, as it does not have a large attack surface. Also, phishing attacks would be more difficult to execute, as the device authenticates the service and checks the host name and includes the end user in the authentication process. The device could even be extended with more functionality to give you even more benefits from emerging attacks in the future.

One of the challenges for the OffPAD is that it has to know what service it should be giving the credentials for, and it must be able to authenticate

[1] http://www.lucidman.org

all such sites. If this is not done for all requests, the device would be open for attacks that could trick the device to hand out credentials to an attacker. How the service authentication is handled is the focus of this Master's thesis.

Server authentication protocols and Public Key Infrastructures (PKI) are already in use, for instance based on the X.509 standard which is the main authentication regime in HTTPS. The X.509 standard has been long used for online, secure transactions for instance by online banking. In the last years, however, we have seen the vulnerabilities of the standard, as attackers have been able to get copies of certificates that could be used to authenticate any service for any browser, letting it believe that it is the authentic service. Successful attacks misusing X.509 has shown that we are in need of better server authentication standards. One proposal for replacing the X.509 is through storing the service' certificate in DNS and protect the data by DNSSEC.

New authentication protocols create stronger server authentication online. It would be beneficial for an OffPAD to also make use of this new authentication standard to avoid some of the same vulnerabilities as exists for web browsers using X.509. The challenge for the OffPAD and the motivation for this Master's project is how the new PKI could be used in an offline environment, and for authenticating data that is not intended to be sent to the OffPAD, but to the client that is actually executing the transactions. Depending on how wide the usage of the OffPAD should be, it should be considered if the OffPAD should be able to fall back to server authentication through the X.509 standard.

Chapter 2

Structure of the thesis

This thesis first looks at the background and motivation for why the problem with offline server authentication is relevant and why a solution is needed. This is followed by the theory around what is needed for the proposal of the thesis, principles of message authentication and details about the different authentication regimes and public key infrastructures and their standards, including the current PKI that uses X.509.

Security Usability is a central part of the Lucidman project. The basic principles behind security usability is covered to be able to decide how the thesis' proposal should be designed. It is important that the service authentication is implemented in a way that is usable for the end users.

The theory part ends with details about the standard and protocols that is needed for the proposal in how the OffPAD should authenticate services. This includes the necessary details about TLS, DNS, DNSSEC and how certificates could be stored and used in DNS.

The second part of the thesis is the main part and is about the project and its proposed solution. It starts with a description of how the project is planned and what methods are used throughout the work with the thesis. The second chapter is covering the Lucidman project in general, to be able to put this thesis in a larger context. The rest of the chapters contain the theory and discussion around the solution this thesis focuses on.

In the conclusion, the main results from the project are reviewed, together with a suggestion for future work.

Chapter 3

Terminology

Some definitions are used throughout in the thesis, and would need some explanations:

- **Service** and **Server** is in most situations referred to as the same. I prefer to use *service* as one *server* might contain more than one *service*, and we want to authenticate the specific service that we are using. Services might be separated on different ports at the same host name, but it could also be separated by its full URL, for instance two different services under `app.uio.no`, served by the same server.
- **User** and **End User** is both referred to as the *physical person* that handles the authentication device and executes the transactions.
- **PKIX** is referred to as the PKI regime that makes use of the *X.509* standard. When the X.509 standard was adopted by the *Internet Engineering Task Force (IETF)*, it was referred to as *PKIX* [10].
- A **Resource Record** is by the *Internet Engineering Task Force (IETF)* referred to as *RR*, which is often used in this thesis where suitable.

Chapter 4

Background

HTTP traffic between server and client is normally encrypted through the use of Secure Socket Layer (SSL), and its successor Transport Layer Security (TLS). For such a communication to be considered secure, the web browsers have to authenticate the server. The server authentication in standard web browsers is based on validating the server's certificate by checking that it has been signed by a third party Certificate Authority (CA). This regime is called «Public-Key Infrastructure using X.509» (PKIX) [10].

The confidence in PKIX have suffered from criticism. The Electronic Frontier Foundation (EFF) says

> The certificate authority system was created decades ago in an era when the biggest on-line security concern was thought to be protecting users from having their credit card numbers intercepted by petty criminals. Today's Internet users rely on this system to protect their privacy against nation-states. We doubt it can bear this burden. [41]

There are different problems with PKIX which have been described in various papers, for example in RFC6698 [19] and «Trust extortion on the Internet» by Audun Jøsang [21]. Some of the challenges are articulated as follows:

- Every CA is able to sign certificates for any domain, so if one CA gets untrustworthy, it breaks the whole PKI. One example is the break-in at Comodo in March 2011 [4] and at DigiNotar in August 2011 [5], where fake certificates were created for e.g. google.com, mozilla.com and torproject.org[1]. The companies have both been accused for keeping the information about the intrusions to themselves too long to avoid going bankrupt, which is not helping on the trustworthiness.

[1]The Tor project is developing software for surfing the web anonymously, and is used in countries like China and Iran to prevent eavesdropping by governments.

- There are only ad hoc solutions for communicating changes to CA certificates. This is why browsers come pre installed with numerous CA certificates, even outdated ones for backwards compatibility. [22]

- PKIX only authenticates the *domain name* of a server, but not its IP address. A missing link between the host name and IP address makes the client open to attacks where a fake or compromised DNS server could hand out fake IP addresses before a real DNS server had the time to reply. A local attacker could even hand out fake MAC addresses, telling it is your intra net's gateway, and then controlling the client's flow of data. Since PKIX only checks the domain name, you would under such attacks not be able to determine if you're talking with the correct server, as long as the certificate received is a "valid" certificate, signed by a CA your client trust.

- The server authentication process does not involve or help the user, which makes it hard for the user to understand that he or she communicates with the authentic service. This problem is exploited by phishing attacks today.

The problems with PKIX, which has received a greater focus in the media in the last years, make entities having less trust in the traditional browser PKI.

There have been proposed different solutions for the PKIX issues. Lately, as DNSSEC has been implemented, people have been interested in using DNS with the trust of DNSSEC to store authentication data for hosts. There are already RFC standards for how this should be done for different protocols, for instance for HTTPS [19]. Users *should* be able to use this standard online today.

However, when using an OffPAD, there is no existing solution to the authentication of web servers through the TLS communication. The offline device is not directly part of the browser's TLS setup, and it will not be able to ask DNS for data either. These are challenges that have to be solved for the OffPAD and the online part that it communicates with. It is especially important that the chain of trust goes all the way from the OffPAD to the server, as we should not trust the client. The authentication of web server transactions has to be done through the OffPAD itself.

Chapter 5

Theory

5.1 Cryptographic algorithms

There are various algorithms that are used in cryptography to support the security primitives and services. This section describes the principles of the algorithms that is referred to in the thesis.

5.1.1 Symmetric and asymmetric encryption

One could categorize the encryption methods into symmetric and asymmetric encryption. In symmetric encryption the sender and receiver are sharing a key that is used to both encrypt the data before sending it and decrypt it when it has arrived. Most symmetric encryption algorithm says nothing about how you should get hold of the initial key, or keys, securely, except that it should be done in an *out-of-band channel*. Asymmetric encryption, on the other hand, uses the principle of private and public keys, and is also referred to as public-key encryption. You would need two keys, one for encrypting the data and another one for decrypting it. The encryption key would not be able to decrypt the data, and vice versa, which is making it asymmetric.

Symmetric encryption has in general simpler algorithms and is more effective in terms of processing power. Asymmetric encryption algorithms are often more complex and require more resources to process. To get the best from both encryption worlds, algorithms typically use an asymmetric algorithm to connect, authenticate and transfer a shared, secret key, before switching over to a faster symmetric encryption by using the shared, secret key.

Public and private keys

Asymmetric encryption depends on a public and a private key, where any one of them could encrypt a message, which could then only be decrypted by the other key. The public key could be given to anyone that wants to give

you data securely, while the private key must be kept hidden from others than the rightful owner. The public key could even be published for anyone to see.

An advantage of public-key cryptography is that it could not only be used for confidentiality and integrity, but it could also be used for authentication. If a message is sent by using the keys the other way around, you could encrypt the data by your private key, and only your public key would be able to decrypt it. This way, the receiver knows that it must be you that sent the given data, as long as the receiver already trusts your key, that is. This mode of operation does not offer confidentiality, only authentication, but confidentiality could be implemented by re encrypt the already encrypted data with the receiver's public key.

5.1.2 Secure hashing

For authentication and integrity purposes, you would often rely on secure hash functions. Secure hash functions are one-way functions that take a message as input to generate a short output value. The difference between regular hash functions and secure hash functions is the algorithm's effort into making sure that it is a irreversible process. You should not be able to reverse the secure hash algorithm and get back to the original message.

The output of the secure hash function is often encrypted by the sender's private key. The receiver is then able to decrypt the cipher text to get the sender's hash value, and could compare it with the hash value generated on the receiver's side. By combining secure hashing and public-key cryptography, you have both integrity and authenticity.

Secure hashing functions could be used to generate values that are sent together with the message, as could also be in plaintext format. The receiver could then generate a hash value out of the message on its side, and compare it with the received hash value. This is faster than encrypting the whole message, and offers authentication and integrity, but no confidentiality.

Another usage of secure hashing functions is to be able to verify that two entities know the same shared secret, without having to send the secret in plaintext. If one entity generates a hash value out of the secret and sends it to the other entity, the receiver could then easily verify that the hash value could only be generated by the correct secret. This is often used when verifying a password.

Secret hash functions supports the security primitives:

- Integrity - The given hash value would not match the receiver's generated value if the data was modified on its way, either by mistake or by an attacker.

- Authentication - The message could be prepended or appended by a secret that is shared by the sender and receiver, before the hash value

is generated. This way, the given hash value would only match the receiver's generated value if the same secret is used, which the receiver then could use to verify that the message is from the correct sender.

- Proof of knowledge with confidentiality - The hash value output of a message could be published without anyone being able to find out about the message. Those that already have the message could verify that the hash value matches the message.

One generic function for generating an authentication hash value of a plaintext message is called *Hash Message Authentication Code (HMAC)* [30], but which is using symmetric encryption. The sender and receiver share the same secret, which is appended to the message before creating a hash value. The receiver should do the same to the received message and try to match it with the given hash value.

5.1.3 Digital signatures

A side effect of public key encryption is digital signatures. As only the sender's public key could decrypt an encrypted message from the sender, you also know that only the sender could have encrypted the message in the first place. The message could then be looked upon as being *signed* by the sender, where the encrypted message is the signature of the original message. A digital signature supports three security services:

- *Authentication*, as the recipient could check that only the sender could have created the message.
- *Non-repudiation*, as everyone could check that only the sender could have created the message. Non-repudiation depends on authentication, with the addition of a universal proof, not only for the recipient[1].
- *Integrity*, as the message must match the given signature for it to be valid. No one could modify a signed message without getting hold of the private key.

Replay attacks could be prevented in various ways, for example by adding a time dependent variable to the message that should be digitally signed. When a time dependent variable has expired, you could no longer accept the signature as valid. Such functionality is implemented in various protocols using digital signatures.

For efficiency and to save storage, digital signatures are most often used by only encrypting an outcome of the original message, to which secure

[1] A shared secret key could for instance be used for authentication for one of the recipients, but it would not support non-repudiation, as nobody else could verify whom of the key holders that had signed it.

hashes are of practical use. By encrypting a hash of the message, you would have a shorter, but still valid, signature, which could be delivered together with the message. [42]

How a signature is generated varies between protocols, for instance what hashing algorithm to use, and what data to hash. Various security related protocols are strongly dependent on the use of digital signatures, for example the X.509 certificates.

5.2 Security Usability

The term *Security Usability* is a too often neglected field of study where the usability of security systems are in focus. The usability aspect of security is often crucial, as user's behaviour could lead to quite bad consequences. The users must be able to understand the system enough to be able to make secure decisions. [23]

In *Security Usability* all user involvement is categorized into two types of involvement, *Security Actions* and *Security Conclusion*. [24] These involvement types have been defined with *Usability Principles* to them:

1. *Security Action* - When the user is actively doing something, like entering a password or pressing a browser button when asked for confirmation about an untrusted website. The Usability Principles:

   (a) The user must understand which security actions are required of him or her.

   (b) The user must have the knowledge and ability to make the correct security action.

   (c) The mental and physical requirement of the security action must be tolerable.

   (d) The mental and physical requirement of making security actions repeatedly, for different services, must be tolerable.

   A user would have less mental capacity when something is happening there and then, like when doing a transaction with a website, compared to slower actions, like setting up the configuration of client, where the user have to look for more information before concluding. Time matters.

2. *Security Conclusion* - When the user is observing something that is relevant to the security, like checking if the URL started with *HTTPS* to verify that the communication is going through TLS. The Usability Principles:

(a) The user must understand the security conclusion required for making an informed decision, and must understand the requirement to support a secure transaction.

(b) The system must provide the user with enough information for the user to be able to derive the security conclusion.

(c) The mental requirement for deriving the security conclusion must be tolerable.

(d) The mental requirement for repeatedly deriving the security conclusion, for different services, must be tolerable.

Even if a security system have a strong cryptographic security, it is of no use for the end user if the security usability is not in place. A common example of missing security usability is when a web browser is asking the user if it should trust a certificate that could not be authenticated by any of the Certificate Authorities certificates already installed in the browser, without giving understandable information to the user. A normal user is not capable of determining if a certificate is authentic or not, without getting more information and verification tools, thus breaking principle 2C. The user *could* then accept the new certificate, just to be done with the transaction, without understanding that he or she could now communicate with the wrong server.

Phishing attacks are often exploiting various security usability flaws to attack users.

5.3 Authentication

Authentication is a basic security service [9] and is one of the fundamental security primitives [27]. Authentication is often put in the same box as access control, integrity, confidentiality and non-repudiation. One definition of *authentication* is *the process of determining whether someone or something is, in fact, who or what it is declared to be* [2]. More technically speaking, authentication could be defined as *the process of determining if an attribute of an entity is true*[3]. In other words, the authentication process should validate one or more attributes of an entity to be able to make sure that the entity, or entity's attribute, is from who it claims to be. The entity could be an individual or something else, like a service, and we must make sure that only the correct entity could have the attribute that we are validating.

Authentication is required for other security services to be usable. For confidentiality you are often dependent on checking the identity of the entity before handing out confidential data. In access control, you must know *who*

[2] searchsecurity.techtarget.com/definition/authentication, seen 7th of January 2013

[3] en.wikipedia.org/wiki/Authentication, seen 7th of January 2013

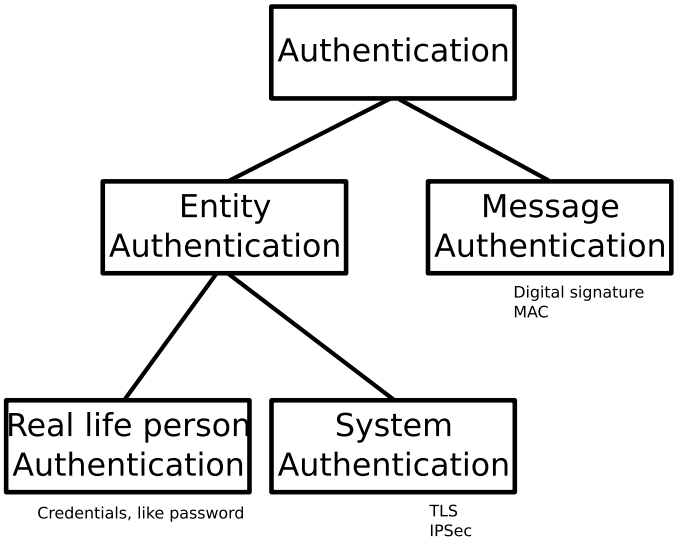


Figure 5.1: The general types of authentication

you should grant or deny access to before actually doing it. Auditing is often dependent on knowing who it is auditing.

The standard «Security Architecture for Open Systems Interconnected», commonly named X.800 [9], separates authentication into two kinds, depending on the target of authentication:

- *Peer Entity Authentication* - direct authentication. The process of confirming if the entity is the one that it claims to be.
- *Data Origin Authentication* - indirect authentication. The process of confirming the source entity of any given data. This is also referred to as *message authentication.*

The types of authentication are illustrated in Figure 5.1.

Mutual authentication is when both parties, both the server and the client, authenticate the other for the same transaction. The focus has often been on protecting the servers and services, having the server authenticate the client side of a transaction, with less focus on the server authentication. [27] If a system only authenticates the client, you would in effect only have confidentiality between the client and some server, as the client could give away information to a fake server. [44]

The processes in server and client authentication could vary, even in the same, mutual authentication process - for example could the user be authenticated through a password, and the server through a certificate.

5.3.1 Peer Entity Authentication

In *Peer Entity Authentication* you need to verify that an entity you are communicating with is the authentic entity as it has *identified* itself as. The authentication type could be further separated by the entity type that is authenticated. The entity types are:

- Legal and cognitive entities, like real life persons and organisations.
- System entities, like servers and clients.

This is an important separation when considering security usability.

> In reality, client and server systems are only agents for legal and/or cognitive entities such [as] persons or organizations. [27]

which means that you could consider the end user entity to consist of two entities, the real life end user U and the user's client software C, and the server entity to consist of the server software S and the Service Provider organization or person O. This is an important distinction when considering the whole chain to protect, as the real life person could often be the weakest link, not the system.

In [27, 44] the different authentication types between the client side and the server side has been discussed. For the client-side authentication:

- $\mathbf{U} \rightarrow \mathbf{O}$: User ($U$) authentication by the SP organisation O.
- $\mathbf{C} \rightarrow \mathbf{O}$: Client ($C$) authentication by the SP organisation O.
- $\mathbf{U} \rightarrow \mathbf{S}$: User ($U$) authentication by the server S. This is often referred to as the *user authentication*.
- $\mathbf{C} \rightarrow \mathbf{S}$: Client ($C$) authentication by the server S.

The server-side authentication:

- $\mathbf{O} \rightarrow \mathbf{U}$: SP organisation ($O$) authentication by the human user U.
- $\mathbf{S} \rightarrow \mathbf{U}$: Server ($S$) authentication by the human user U. This if often referred to as *Cognitive Server Authentication*.
- $\mathbf{O} \rightarrow \mathbf{C}$: SP organisation ($O$) authentication by the user's client C.
- $\mathbf{S} \rightarrow \mathbf{C}$: Server ($S$) authentication by the user's client C.

These types of authentication are all shown in Figure 5.2 on the next page, from the paper *Service Provider Authentication Assurance* [27] and updated in *The OffPAD: Requirements and Usage* [44]. To have a successful, mutual, end-to-end authentication, you would need the authentication of type $\mathbf{U} \rightarrow \mathbf{S}$ and $\mathbf{S} \rightarrow \mathbf{U}$ to succeed, as shown in Figure 5.3 on the following page. Authentication between the server S and the user's client C would not give you a *cognitive* server authentication. [44]

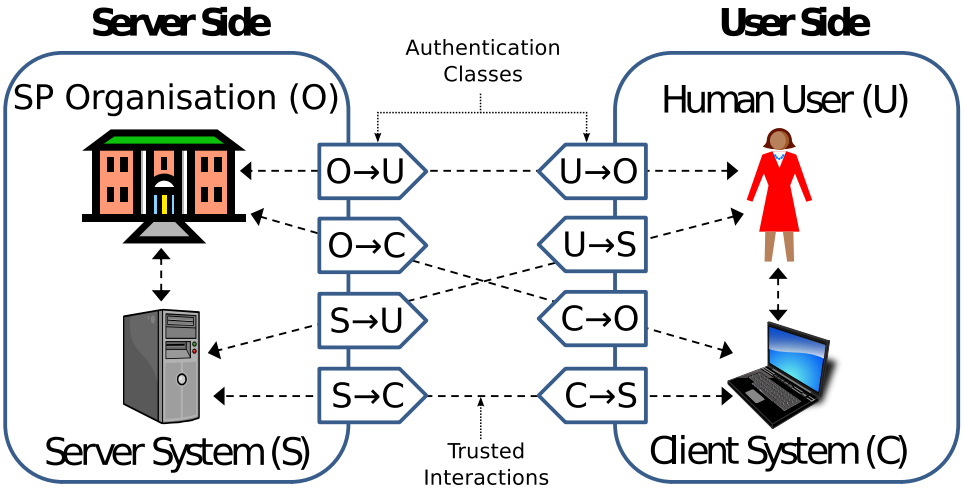


Figure 5.2: The types of mutual authentication [44]

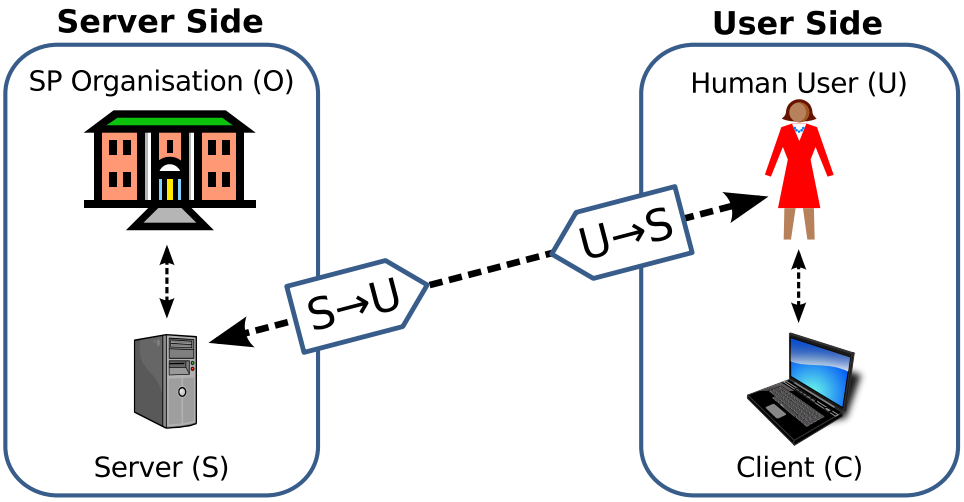


Figure 5.3: Mutual and cognitive end-to-end authentication for an end user and a service

5.3.2 Message authentication

In message authentication, you need to verify that the issuer of given data or a message is generated by the claimed source. This is important both in systems and real life, as messages delivered by other entities could affect a person's or system's decisions. Message authentication could be an offline process, as the message sender is not involved in the process, depending on the authentication process.

X.800 separates *Data Origin Authentication* from *Data Integrity* in two separate functions:

> The *data origin authentication* service provides the corroboration of the source of a data unit. The service does not provide protection against duplication or modification of data units. [9]

Instead, *Data integrity* may be used when sending data after *peer entity authentication*:

> On a connection, the use of the peer entity authentication service at the start of the connection and the data integrity service during the life of the connection can jointly provide for the corroboration of the source of all data units transfered on the connection, the integrity of those data units, and may additionally provide for the detection of duplication of data units, e.g. by the use of sequence numbers. [9]

The authenticity of a message would still be meaningless if it had no integrity verification.

5.3.3 Authentication modalities

The authentication modality, i.e. the strength of the association between an entity and its claim of authenticity, is in [26] split into three modality types, *syntactic authentication*, *semantic authentication* and *cognitive authentication*.

- **Syntactic entity authentication**:

  > The verification by the relying entity that the identity of the entity in an interaction is as claimed. [26]

  This is the type of authentication as defined in X.800 [9], and does only provide authentication between systems, but does not provide much help for the human user. For instance would it be easy for users to not be able to see that `www.paypal.com` and `www.paypa1.com` is two different addresses, even though both could be fully authenticated by the web browser.

- **Semantic entity authentication**:

  > The verification by the relying entity that the identity of the remote entity in an interaction is as claimed, and in addition the verification by the relying entity that the remote entity has semantic characteristics that are compliant with a specific security policy. [26]

  The difference between semantic authentication and syntactic authentication is that with semantic authentication only *some* entities could be automatically authenticated for a given purpose, while all entities *could* be authenticated with syntactic authentication.

- **Cognitive entity authentication**:

  > The verification by the cognitive relying party that the identity of the remote entity in an interaction is as claimed, and in addition the verification by the relying entity that the remote entity has semantic characteristics that are compliant with a specific security policy, and in addition the appropriate presentation of semantic characteristics of the remote entity in a way that makes a cognitive relying party able to examine the true nature of the remote entity and to judge policy compliance. [26]

  With this authentication modality you include a *cognitive party*, the human user, in the authentication process. For the entity to be fully authenticated, the user must be given enough data and tools to be able to fully authenticate the entity without any doubt.

  Cognitive entity authentication gives a stronger authentication, but requires more of the system that implements it, especially considering security usability.

5.3.4 Authentication methods

In the security context, the authentication process must be done in a way, so that attackers are not able to masquerade as others. This is often done by implementing thorough, mathematical algorithms which are computationally impossible to solve if you do not know the secret that the authentic entity knows.

There are different ways to authenticate messages and entities:

- By using public keys for signatures.
- By using a shared secret that both entities knows. User name and password is the common example of a shared secret.
- By trusting a third party.

Human authentication

Your brain can not compute as good as computers, it can for instance not create digital signatures by itself. Instead, in personal authentication you depend on other properties, as described in the *the NIST Handbook* [17]:

- Something you know, for instance a password or a PIN.
- Something you have, for instance a smart card or a code generator.
- Something you are, for instance your fingerprint or iris.

When a system requires two-factor authentication, it typically combines «something you know», the password, with «something you have», like a paper with a list of one time codes, or a device that generates one time passwords. [44]

Machines would normally authenticate through using *something they know*: computing mathematical problems that are easy to calculate when knowing the secret, but which requires a lot of processing power to solve if the secret is not known.

Public key authentication

By using public keys, an entity could create a signature on any data. This could be used to both sign a message that needs authentication, but a signature could also be used to authenticate the entity itself. The sender adds a digital signature to the message. This is done differently from protocol to protocol.

The data and signature could be verified by using the sender's public key, which requires that the receiver already has this public key.

Systems that makes use of this authentication method, are systems that makes use of X.509 certificates, like TLS. Other systems using this are PGP and SSH.

Shared secret

Another way of authenticating a message or an entity is through validating a secret that both the sender and the receiver should know about.

The secret is normally not transferred in plaintext, but is protected differently in various protocols:

- Through using Message Authentication Codes (MAC), like the hash-based message authentication code (HMAC): The sender creates a hash out of a message and the shared secret. The receiver then tries to generate a hash out the given message and its own version of the secret, and checks if it matches with the given MAC.

- By adding confidentiality to the message, e.g. transferring the secret through a secure transport like TLS. This is a common way of authenticating users' passwords in HTTPS, as it gets sent in plaintext, wrapped inside of TLS.

  This is normally used for entity authentication. It *could* be used for message authentication, but it would require that the message gets encrypted by other algorithms.

Trusted third parties

Trusted third parties could take care of the authentication of entities. It would require the receiver to trust the third party, and it would also require the receiver to be able to authenticate the third party, or that the communication with the third party is protected, e.g. through an out-of-band channel. This is how many *PKI systems* work.

5.3.5 Authentication phases

To authentication other entities successfully, you are required to support different phases. This should not be confused with the processes in a PKI system in 5.4.1 on page 28, which extends this. The phases are:

1. *Registration phase*: For others to be able to authenticate an entity, it must first be registered in such a way that the receiver is able to authenticate it. The entity must be identified and registered, either online or in the real world, depending on the authentication system. The authentication information needs to be provisioned to the receiver(s). One example of this is public keys, which could be delivered to the receivers, either by meeting them in person, or simply by publishing them online.

2. *Operational phase*: When the entity is registered, the normal identification and authentication process could be executed.

3. *Termination phase*: Authentication data must be cleaned up properly. An entity's authentication data could be revoked for various reasons, e.g. if the entity does not exist anymore or it has been compromised.

Initial authentication

The first step for authenticating an entity is often hard to solve. How could you make sure that an entity is the claimed entity if you have never authenticated it before and have nothing to compare with? You might not have any public key to compare with, and if you are using a shared secret for authentication, you need to authenticate the entity before you share the

secret with it. Even when you have the entity's public key, you must be sure of the authenticity of that key.

The initial authentication process is solved differently in systems. Some ignore the problem, and blindly trust the initial attributes of an entity, while others solve it by handing the responsibility to the end users by telling them to get the attributes in «*an out-of-band channel*». The initial authentication phase is a crucial part of many PKI systems, as this is something they try to solve in a way that is usable for the end users.

Even when using PKI systems, you also have the challenge of the initial authentication of the PKI system itself. How could you trust the PKI system's entity if you have never authenticated it before?

You must initially have to trust *something* for authenticating entities, independent of what authentication protocol you are using. All the authentication steps need to be verified, even those before the process of authenticating an entity itself.

Trust On First Use (TOFU)

The term *Trust Upon First Use (TUFU)*, or *Trust On First Use (TOFU)* refers to security algorithms where client software is directly or indirectly trusting a service when connecting to it for the first time. After the initial authentication, the client already has the correct authentication attribute of the service, and could therefore authenticate the service more properly. The client would abort the communication if a service' attribute has changed since the initial authentication. This principle is solving the initial authentication process by simply ignoring it.

One example of a client that practices *TOFU* is the *Secure Shell (SSH)* [45]. In SSH, when connecting to an SSH server for the first time, the user gets asked:

```
user@localhost ~ $ ssh someotherhost
The authenticity of host 'someotherhost (10.0.3.43)' can't be established.
RSA key fingerprint is e2:f0:7d:3f:9b:1f:3d:1a:1a:6d:3b:90:10:01:12:1c.
Are you sure you want to continue connecting (yes/no)?
```

Since the client does not know if this data is true, it is up to the user to verify if the fingerprint is correct. The lazy user could simply accept the connection, and the certificate would get stored in the SSH client's list of known hosts to be accepted when reconnecting to the service later.

When the user is connecting to the server the next time, and the same certificate is used, no prompts would be given, and the user could get asked for its credentials. However, if the server's certificate has changed, the client would be given a message:

```
user@localhost ~ $ ssh someotherhost
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx.
Please contact your system administrator.
Add correct host key in /home/username/.ssh/known_hosts to get rid of this message.
Offending key in /home/username/.ssh/known_hosts:1
RSA host key for someotherhost has changed and you have requested strict checking.
Host key verification failed.
```

Which means that the user would have to do something more actively. A quick solution for the lazy user is to simply remove the previously authenticated fingerprint from the list of accepted servers, which would make the SSH client redo its TOFU algorithm.

Another example on the TOFU principle, is the HTTP header attribute *HTTP Strict Transport Security (HSTS)*, which was finally standardised in RFC6797 [18] in November 2012. This header should be given to the client to tell it that it should *only* communicate with this server through *HTTPS* and not *HTTP*. This is used to e.g. prevent SSL-stripping, where the attacker is blocking the *HTTPS* reply from the server and returns an *HTTP* response instead, and from there being able to both eavesdrop and modify the transactions. By using *HSTS*, the client would not accept *HTTP* responses, but *this only works if the client has communicated with the server before*, and has already received the HSTS header. If the client has not communicated with the server before, it is still trusting the server even if it changes over to *HTTP* communication. Some web browsers have created their own, internal lists of domains that should use HSTS even at the initial request, to prevent attacks on the initial request.

5.3.6 Revocation

The term *revocation* is the process of revoking a previously given attribute. In a security perspective, we are talking about the revocation of authentication attributes by making them invalid. Revocation is the process of revoking the grant, and the process is different depending on the system and how the authentication process works in the system. In online systems were the user logs on access is revoked by simply disabling the account from the system.

The revocation process when using certificates for authentication is more complicated, as the certificate is already given, and you can not just «take it back». The certificate could even have an end date set many years in the future. The solution for checking if a certificate is revoked is therefore to ask an online service, e.g. administered by a CA. One standard for revocation lists is the *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, which defines how revocation should work for X.509 certificates. The revocation itself could be given

by CAs in a signed, time-stamped list, where each revoked certificate is identified by its serial number.

The online checking of blacklisted certificates has had many problems, it is not scalable or even economically justified for the CA companies to scale up. Today, when certificates are compromised, e.g. when CA companies have been attacked, the largest browsers has updated their own hard coded blacklist of *certificate fingerprints*[4] by the developers. This tells us that revocation services are not trusted by the browser developers for preventing misuse of certificates.

5.3.7 Authentication threats

Attackers have often a motivation in money and/or reputation for successfully attacking a system or a user. If the authentication process in a system breaks, the attacker have the possibility to act as a different entity, masquerading, often with the same privileges as the entity. By masquerading, the attacker has a lot of potential of misuse of a system, often also without being noticed. Examples are transferring money to the attacker's account, or getting access to confidential, valuable data. The authentication process is a lucrative attack surface, often because it's the most visible surface to a system, and because of the potential in breaking it.

5.4 Public Key Infrastructure (PKI)

«The PKI is a set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.» [43, 42] The purpose of a PKI is to manage the binding of public keys to end entities in a practical way that is still ensuring this binding securely. The most important part of any public-key system is the *end entity* which are to be authenticated through its public keys stored in certificates. PKI systems often depends on one or more *trusted third parties* to make the authentication procedures manageable in large scale infrastructures. The types of trusted third parties and their purpose:

- *Certificate Authorities (CA)*, which are issuers and revocators of the certificates for the end entities. A certificate would have a binding between a target entity and the CA that created and signed the certificate.
- *Registration Authorities (RA)*, which could manage the registration process of end entities on behalf of the CA. The CA could do this job itself.

[4] A *certificate fingerprint* is a hash out of given data in the certificate, like the *Common Name*, signature and issuer.

- *Validation Authorities (VA)*, which would provide the authenticated information to entities on behalf of the CA. The CA could do this job itself.

- *Revocation publisher*, also known as *CRL issuer*, which could publish lists of revoked certificates on behalf of the CA. One format of a revocation list is the *Certificate Revocation List (CRL)*, but other formats exists.

5.4.1 Processes in a PKI

Some processes that could occur in a PKI [42]:

- *Registration process*: The end user communicates with an RA that is delegated the administrative task of a CA. The RA has to identify and authenticate the end entity, either off-line or through some procedure that would verify the authenticity of the entity, before the entity could get a certificate.

- *Certification process*: The certificate gets generated by the CA and given to the end entity. The entity have to be authenticate before this step could occur. The initial entity authentication varies, even in the same PKI[5]

- *Authentication process*: One entity needs to authenticate another entity. For the entity authentication to happen, the initial entity gets the certificate for the unauthenticated entity and verifies that the public key in the certificate could only be generated by the real entity. If a VA is involved in the process, the initial entity would need to trust the VA or the certificate's CA, and would also need to authenticate the VA and/or the CA. Often, the initial entity would also like to check if the given certificate has been revoked.

- *Revocation process*: The process of making a certificate useless. A revocation would be needed in situations where an end entity's private key is compromised, or the entity has changed its attributes or simply left the domain of the PKI, typical a company that has resigned. The CA has to be notified of such situations in a authentic manner, before it could revoke the certificate.

Some PKI systems does not depend on trusted third parties, CAs, as the end entities would then need to authenticate the other entities themselves. An example of a PKI without a CA is the PGP [8] implementation, where

[5] In X.509's basic certificates, without any certificate extensions with later standards, it is up to the single CA to decide what is good enough to be initially authenticated, varying from e-mail to physically presence.

the users have to authenticate other entities by their own or through other entities they already have authenticated and trusts. In PGP, it is up to the end users themselves to define how the authentication process have to occur before an entity could be marked as authenticated. The certification process is then using the user's own, self signed key to sign the end entities' public keys.

5.4.2 PKI categories

The types of PKI systems could be put into three categories:

- *CA usage*, where each entity authenticates other entities through certificates generated by trusted third parties, the CAs. The typical PKI in this category is the PKIX, using the X.509 standard.
- *Web of Trust (WOT)*, where each entity is authenticating other entities by itself. In addition, you can mark some of your own entities to be trustworthy, and therefor also trust their list of authenticated entities. When authenticating your entities' list of entities, you have your own network of trust inside of the PKI. One such PKI is the *PGP* system.
- *SPKI*, which is really a specific PKI standard [13], but could be put into its own category. In SPKI, you only have a binding between a public key and its authorizations, instead of binding an entity's name to the public key. A CA serves no purpose in SPKI, as the delivery of the certificate should be done directly between the key holder and the verifier, thus being most practical in small scaled and closed systems.

5.4.3 PKI systems

Some of the common PKI systems:

- X.509, the current standard used online, which is using trusted third parties for generating and signing certificates for entities.
- TLSA, a newly defined standard, using DNS to store public keys for hosts. More information on this standard is found in Section 5.7.1 on page 42.
- Pretty Good Privacy (PGP), a standard for encrypting, decrypting and signing data with the use of public and private keys. The standard is defined in RFC4880 [7]. It does not have a global PKI, but the entities, or enterprises, could manage their own PKI, and decide whom they trust.
- Simple Public Key Infrastructure (SPKI), an experimental proposal for a new PKI. It was defined in 1999, in RFC2692 [13] and RFC2693 [14]

with the motivation of creating something less complicated than X.509. It does on purpose not use commercial Certificate Authorities.

5.4.4 Public Key Infrastructure using X.509 (PKIX)

The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) defined a standard for Public Key Infrastructure (PKI) called X.509[6], which was later adopted by the Internet Engineering Task Force (IETF) and is now defined in RFC5280 [10], and is often referred to as *PKIX*. It was initially published in 1988, but the RFC specification was last updated in May 2008. Later, a working group in IETF named *the PKIX Working Group* has produced more RFCs with updates to the standard to support more needs for usage on the Internet.

The PKIX specification is the PKI system that is used in browsers today for authenticating web sites when using HTTPS, and is therefore probably the most known PKI for most end users. Most online users have had to make a decision in their browser due to X.509 certificates.

X.509 certificates

A sentral part of the X.509 standard is the digital certificate format [10]. The X.509 certificates are strongly dependent on digital signatures for making the certificates authentic. An X.509 certificate contains two parties, the issuer of the certificate, and the subject the certificate is giving information about. The main message in an X.509 certificate is the subject's public key, but it also includes various meta data about the subject and issuer.

Some of what an X.509 certificate contains:

- *Version* of the X.509 certificate format, which is today ranging from 1 to 3.
- *Serial number*, which must be unique within the issuer's certificates.
- *Valid period*, which tells you when the certificate starts and when it expires.
- *Algorithm* used for the digital signature, for example RSA.
- *Issuer name* of the entity that generated the certificate. The name must follow the X.500 standard [20].
- *Subject name* of the entity that the public key in the certificate belongs to. The name must follow the X.500 standard [20].
- *Subject's public key*, which is the main message in the certificate.

[6] http://www.itu.int/rec/T-REC-X.509

- *Extensions* of the certificate, that is for various usage.
- *Signature* of all the other parts of the certificate, by using the issuer's private key.

Note that the certificate does not contain the issuer's public key. That needs to be retrieved through other channels. Most often, the issuer's public key is stored in another certificate, either signed by another issuer or by the subject itself.

The X.509 certificates support a hierarchy, where one certificate issuer could have a certificate issued by another issuer higher up in the certificate hierarchy. On the top of this chain, you could have one or more root issuers, as these would not have a certificate issuer, but instead, their certificates are signed by the same entity as the certificate's public key is from, themselves, also known as *self signed certificates*.

It is possible for one public key to be signed by different issuers, which means that you could have many certificates for the same key. This could complicate revocation processes [16].

The trust model

The infrastructure's trust model is based on Certificate Authorities (CA), which signs web site owners' public keys with their own private keys. The CAs' public keys are all pre stored in the web browsers, and might vary from what the responsibility of the browser decides.

The trust model is idealistic in a hierarchy, where the CA is on the top, and delegates trust to other sub-CAs, and where the leaf nodes are the entities that wants to be authenticated. A browser that receives a service' certificate, checks the issuer of the certificate. If the issuer of a certificate is in the browser's list of trusted CAs, the service gets accepted. If the issuer's certificate is not a trusted CA, the browser would check the issuer of this certificate. The browser would check the issuer of all certificates, until it either finds an issuer which is trusted, and accepts it, or the chain stops at a certificate that has already been checked, and denies it.

Disadvantages of PKIX

In reality, the structure of CAs in PKIX is complex. [16] The CA hierarchy is not always top-down. [22] In PKIX, it is also possible for a CA to sign another CA's certificates, which has been used to avoid that some browsers would not be able to authenticate some sites as they did not have the original CA's certificate in their pool of trusted CAs.

Every CA is able to sign any certificate, for any domain. There is no technical limit in PKIX for doing this. This means that if one CA is either successfully attacked or is corrupt the whole system is broken. The DigiNotar

break in in 2011 is a school example of this, where false, but valid, certificates were generated for e.g. gmail.com and google.com, even though DigiNotar had no relationship with Google [5]. Another problem is that the certificates are not validating the web site's IP address, which means that attacker could fake an IP address and be able to give a user a fake, unencrypted web site that looks like the real one.

5.4.5 Pretty Good Privacy (PGP)

PGP is a popular *Web of Trust* system, as each entity is itself responsible for authenticating other entities that it should be communicating with. The term *PGP* could refer to the standard, but it could also refer to the *software* that handles the public and private keys, signatures and making use of them for encrypting and decrypting messages. The term *OpenPGP* is the name of the standard, written by IETF in RFC4880 [7], based on the original functionality in PGP.

Since the entities are themselves responsible for authenticating other entities, there is no need for trusted third party CAs in OpenPGP. Instead, the system provides some help in the management of entities, authentication and trust. An entity A could also flag another entity B for transitive trust, which means that A would automatically accept all entities' authenticated by entity B.

5.4.6 SPKI

Simple Public Key Infrastructure (SPKI), was an experimental proposal for a new PKI, defined in 1999 in RFC2692 [13] and RFC2693 [14]. The motivation of the PKI was to creating something less complicated than X.509.

SPKI does not involve any commercial CA, which makes the PKI not interesting for commercial vendors to sell or distribute, as it lacks a way of income. SPKI is also most beneficial at local usage, as wider use would get more costly for the registration process.

5.4.7 DNSSEC

DNSSEC is a hierarchic structured PKI which is used in DNS for authenticating the DNS data and to prevent network attacks like DNS spoofing. DNSSEC is much more structured and constrained than the PKIX regime, as there is only one DNS root key on the top of the hierarchy which is globally acknowledged.

More information about DNSSEC is put in its own section, at Section 5.6.2 on page 36.

5.4.8 Threats to PKI

There are many threats that a PKI must prevent in order to provide value for end users, especially when online.

Registration process

In all PKI systems, you need to authenticate an entity the first time. When the technical authentication process could be thoroughly documented, the registration process might not be documented with other information that it should be «secure» and «through an out-of-band channel».

Lacking a well-documented routine for the registration process, no matter why it is not done, opens up for attackers to try to register as others. Some CAs that generates X.509 certificates might only require an unencrypted e-mail as enough information to authenticate the entity and respond with a signed certificate.

Fake data

Attackers could fake *any* data that is sent either way, either over the network or through some other channel. An authentication system must be able to prevent that any change in the transmissioned data could lead to a false authentication.

One known way to fake data, is through DNS poisoning. If the DNS server the client is using is poisoned by an attacker, it could give the user the attacker's IP address for a given host name. The host name and IP address would now not be for the same entity, which the authentication system should be able to detect.

This is especially dangerous in initial authentication processes, as the client now knows nothing. To prevent this, the client must know something beforehand, which should have been given in a secure channel, for instance meeting in person.

Collision attacks on the message hashes

Attackers could be generating a message that gives the exact same hash as the original message, thus tricking the clients into believing that the message was signed by the authentic sender. This type of attack is called a *collision attack* and requires quite some processing powers. The attack is most often used in offline authentication processes, as it would take some time to create a proper message.

To prevent collision attacks, the PKI could for instance limit the life time of the message to be shorter than the time it would take to generate another message with the same hash result. The secure hash algorithm must not have security issues that could lead to finding a proper hash faster.

Faking certificates

If the algorithms in the PKI system, or the receiver's software, contain flaws when authenticating messages, an attacker would be able to fake certificates.

There are examples on client software or software that either contains flaws or is designed so that it is easy for the developers and system administrators to set it up in the wrong way, and either not authenticate the sender at all, or not checking some part of the certificate, e.g. if the service are located at the correct location.

Attacking the chain

A successful attack on a part of the chain, e.g. a Certificate Authority, could make the attacker able to authenticate itself. What are the risks of this, and how much would the attacker be able to do in such a scenario?

5.5 Transport Layer Security (TLS)

Transport Layer Security (TLS) is a standard for integrity and confidentiality protected communication over the Internet. It has been defined and updated in different RFCs, with RFC5246 [11] as the latest version, TLS 1.2. It is used in the *Hypertext Transfer Protocol Secure* (HTTPS), defined in RFC2818 [39], where the HTTP communication goes through a secure TLS communication channel to and from the client and the web server.

How TLS works is that it starts with a *three way handshake* for setting up the secure communication. It starts with the client asking for a connection together with information about how it wants it set up, such as what algorithms it wants to use for encryption, authentication and integrity checks, often called the *Cipher Suite*. The server responds with how it have decided the communication should work, and might include one or many certificates if that is needed for the authentication process.[7] It could include more certificates if it wants to recommend some specific CA to be used for authentication, but it is up to the client how it wants to authenticate the server. The client could now, if that is required, feed the server with the client certificate. After both parties have authenticated or accepted the other part, they agree upon a shared key that will be used for the TLS communication. The chosen *Cipher Suite* sets what algorithm to use for generating, and sharing, of the shared key.

The usual way for the client, e.g. a web browser, to authenticate a server is through validating that the server certificate is signed by any CA that comes pre installed with it. Normal browsers have hundreds of such

[7] The Anonymous Diffie-Hellman and Anonymous Elliptic Curve does for instance not require any certificate.

CA certificates pre installed[8]. This is however not defined in the TLS specification, but is up to the policies of the given TLS server and client.

TLS is used differently in different environments. The standard does for instance not decide if the server must feed the client with the chain of certificates up to a CA certificate or not. For HTTP servers, the servers most often returns its chain of certificates. The RFC5246 says:

> One advantage of TLS is that it is application protocol independent. Higher-level protocols can layer on top of the TLS protocol transparently. The TLS standard, however, does not specify how protocols add security with TLS; the decisions on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged are left to the judgment of the designers and implementors of protocols that run on top of TLS. [11]

This has the advantage that TLS has a flexible usage, but the downside is that TLS could be used very differently for online services and means that clients that should comply with most of the services must be able to support the various usage.

5.6 Domain Name System (DNS)

The Domain Name System (DNS) is one of the core parts of the Internet. Its main purpose is to map host names into the hosts' IP addresses, but it can contain other types of data. It is designed in a distributed manner, where different entities administrate their own zones of domains. On the top, the *Internet Corporation for Assigned Names and Numbers* (ICANN) owns the *root node*, which redirects every resolver to the Top Level Domains (TLD) like `.com` or `.eu`, which again forwards the resolver to a sub domain under the given TLD.

A *name server* is the DNS servers that has all the information about a zone. If a resolvers asks a name server for information for a sub zone, it will point the resolver to the sub name server. *Authoritative name servers* have the authoritative information about a zone, but this information can be copied over to other name servers for redundancy.

The clients that asks about data from DNS are called *resolvers*. A *recursive resolver* is a client that is doing all the work to get an answer itself, starting with the root node and going downwards. Every recursive

[8] A check on a computer recently installed with Xubuntu 9.04 had 284 certificates in its cert directory. A computer running Red Hat Enterprise Linux 6.2 at the University of Oslo had 120 CA certificates installed and was last updated April 3, 2010. The Mozilla foundation has a file with its certificates under https://mxr.mozilla.org/mozilla/source/security/nss/lib/ckfw/builtins/certdata.txt

resolver must come pre installed with the IP addresses of the root nodes, as it needs some place to start asking.

Stub resolvers are resolvers that depend on other, recursive name servers to do the work for them, by fetching and validating the data from DNS. This is typically how the resolvers in operating systems works. Typically the home users' Internet Service Providers have a recursive name server that works for all its customers. An advantage with this is that the recursive name server caches the recently fetched data, so the data can be used for many users, and they might get their answers quicker. A disadvantage is that the stub resolver has to trust the recursive resolver to give it the correct information.

5.6.1 Resource Records (RR)

Resource Records (RR) are the data that is stored in DNS, and is returned to the resolvers. Typical record types are *A records*, which contain the IPv4 address to a host, *AAAA records* containing IPv6 addresses and *NS records* that links to name servers that should know more about what was requested. There are also an increasing number of other RR types, like the *SRV record* for storing the IP address to specific services, the *TXT record*, defined in RFC1035 [38], originally used for human-readable strings, but could be used for strings of arbitrary data [40], and the *CERT record*, from RFC4398 [28], for storing certificates in DNS for generic usage.

5.6.2 Domain Name System Security Extensions (DNSSEC)

After DNSSEC had been implemented in DNS servers and clients worldwide, people saw the benefit of its authentication and trust and found it to be a suitable place for publicising information that should be publicly available, but which requires to be authenticated, like certificates.

> The DNS security extensions provide origin authentication and integrity protection for DNS data, as well as a means of public key distribution. These extensions do not provide confidentiality. [1]

DNSSEC implements the use of public key cryptography in DNS to be able to authenticate the information stored in DNS. An important part of DNSSEC is that all data returned by a name server has been signed by its own key, which is again signed by the name server's parent. That way, a resolver can verify the data by checking the name servers public keys, which for instance prevents DNS spoofing attacks. Also, when a name server does not have the requested information, a negative response is given, to avoid attacks where parts of the DNS information gets blocked.

The algorithms used for the keys in DNSSEC could either be RSA/SHA-1, which is mandatory for clients and servers to support, but others are

optional, e.g. DSA/SHA-1 or Elliptic Curve. [3, Appendix 1] It also supports specifying your own formats that is not described in the RFC.

When DNS data is retrieved with DNSSEC enabled, the size of the packet gets larger than the size of an UDP packet, which is 512 octets by default [38]. When name servers or resolvers should send responses that are larger than an UDP datagram, it will instead be given through TCP packets [38] to keep the data in order.

The process of signing and authenticating data from DNS is described in RFC4035 [2]. In RFC4034 [3] the basic RR types for DNSSEC is described. The RR types used further in this thesis is covered here.

The DNS Public Key (DNSKEY) RR

The `DNSKEY` RR contains one of a name server's public key. The name server uses its private key to sign other RR sets, and publishes the public key in a `DNSKEY` RR, for the resolvers to be able to authenticate the RRs.

To ease administration, zones could have more than one public key in use. One *Key Signing Key* (KSK) could be used to sign one or more *Zone Signing Key* (ZSK). The private key of the KSK is stored offline, and is now and then used to sign the online ZSK. The private key of the ZSK is used more often, and is therefore more vulnerable to attacks. The DNS root key is an example on an offline stored KSK that is periodically used to sign a new ZSK for daily use. Only ZSK is used for `RRSIG` RRs.

The `DNSKEY` RR contains the data:

- *Flags*, where the important part is if the key is a ZSK or not.
- *Protocol*, which should only contain the value 3 for now. All other values makes the RR invalid.
- *Algorithm*, which tells what cryptographic algorithm the public key is used in, and the format of the public key. The algorithm `RSA/SHA-1` is mandatory to implement in resolvers, while `DSA/SHA-1` is optional. The algorithm `RSA/MD5` has the status *not recommended*.
- *Public key*, which contains the key in the defined format, encoded in Base64.

The Resource Record signature (RRSIG) RR

A `RRSIG` RR contains a signature for a RR set, generated by the name server's private key, and could be verified by the related public key in the name server's `DNSKEY` RR. This is used by resolvers for integrity and authentication of the given RR set.

The `RRSIG` RR contains the data:

- *Type Covered*, defining what type of RR the `RRSIG` is covering.
- *Algorithm*, which defines what algorithm were used to create the signature. The RSA algorithm is mandatory to implement, but DSA and other algorithms are also available. [3, Appendix A.1]
- *Labels*, which gives us the number of labels[9] in the owner name.
- *Original TTL*, which tells us the original *Time To Live* of the original RR set. The TTL gets decreased when time passes by, and the original TTL is needed to recreate the signature.
- *Signature Expiration* and *Signature Inception* sets the end and start time, respectively, for when the signature is valid. This comes in addition to the TTL of the RR set.
- *Key Tag*, contains the key tag value of the DNSKEY which is used in the signature. This is not a unique identifier of the public key, but is making it more efficient to find the correct public key to use.
- *Signer's Name*, which contains the owner name, i.e. the name of the zone that owns the targeted RR set and which has created the signature in the `RRSIG`. This is to support cache resolvers handing out signed data for name servers.
- *Signature*, which contains the signature itself. The signature is generated out of all parts of the targeted RR set, and also the parts of the `RRSIG` RR itself, except for the *Signature* field. The details of the signature process is defined in RFC4035 [2].

Security-aware resolvers are able to verify that a given RR set has been produced by the authentic name server, by checking that the signature in the `RRSIG` RR matches the given RR set and the `RRSIG` RR itself. Only the authoritative name server is able to create the matching RR set and `RRSIG` RR.

The Delegation Signer (DS) RR

The *Delegation Signer (DS)* RR is used to verify that a child zone's `DNSKEY` RR is the correct key. The `DS` RR is stored in a zone that delegates data to child zones, and the corresponding `DNSKEY` RR is stored and owned by a given child zone. The `DS` RR contains a digest of the corresponding `DNSKEY` RR to cryptographically authenticate the given public key. Since the `DS` RR is followed by an `RRSIG` RR, it could be verified by resolvers. If a resolver trust the given zone's public key, it would also trust the child zone's public key, as the public key could be authenticated by the given zone's signature.

[9] A label is a part of a host name, e.g. contains `sub.example.com.` 3 labels

The DS RR contains the data:

- *Key Tag*, which is a short hand reference for finding the correct public key more efficiently. It does not uniquely identify the key, which the *Digest* value does.
- *Algorithm*, telling what cryptographic algorithm the corresponding public DNSKEY is stored in, and its format. An example is RSA/SHA-1.
- *Digest type*, telling what algorithm is used to generate the digest. SHA-1 is a mandatory algorithm to be implemented by resolvers.
- *Digest*, which is the digest of the corresponding DNSKEY RR, stored in the child zone. The digest hash is generated over the owner name of the DNSKEY and all the content of the DNSKEY RR, that is the flags, the protocol setting, the algorithm setting and the public key itself.

The use of DS RR is DNSSEC's standard way of supporting the chain of trust from one zone to a child zone. The root zone does not have a DS RR to vouch for its public key, which has to be pre fetched and trusted. The root zone authenticates the top level domains that supports DNSSEC by using DS RR for each of their entry DNSKEY RR.

The Next Secure (NSEC) RR

The *Next Secure (NSEC)* RR is used for *authenticated denial of existence* [3], in that it is returned as response to requests for data that doesn't exist in the given zone for which the name server would be authoritative for. The RR contains a name to show that nothing exists between the name of the RR itself and this *next* name. The NSEC RR is followed by an RRSIG RR to prove its authenticity.

The NSEC RR contains the data:

- *Next Domain Name*, which contains the next *owner name* for a name server which has authoritative data, or a delegation point for an RR of the type NS.
- *Type Bit Maps*, which tells what RR set types that exists at the given *Next Domain Name*. Each RR type is in DNS registered with its own integer value, starting at the type A as value 1 and NS with the value 2.

An RR type related to the NSEC RR is the NSEC3, which works as the same way, but to avoid zone enumeration or *zone walking*, i.e. resolvers could get a list of all hosts in a given zone, it returns a salted hash out of the *next* name. [36]

The chain of trust

By using the DNS keys from the root node and down, we get a chain of trust that can be cryptographically verified for every node in the chain. The chain prevents resolvers from being attacked with bogus data from others than the name servers themselves.

Note that every node in the chain has to support DNSSEC for it to be secure. The root node was first signed June 16, 2010 [6], which completed the chain from the top. Some TLDs had already been using DNSSEC without being signed themselves.

Note that not all TLDs are using DNSSEC, either because they are not there yet, or because they do not want to support the extra work for supporting it. Finding out if a TLD is signed is as easy as running `dig TLD. DNSKEY`, for instance:

```
$ dig +short com. DNSKEY
257 3 8 AQPDzldNmMvZFX4NcNJOuEnKDg7tmv/F3MyQROlpBmVcNcsIszxNFxsB f...
256 3 8 AQO+/56uGUHXvOkjGGlaY9IVCOwv55QfC4NXezPHQKg9zexkHifvAHvS c...
$ dig +short no. DNSKEY
$
```

which tells us that `.com` is using DNSSEC, while `.no` is not. This is important to remember when setting up an e-commerce domain, as every name server in the chain has to be signed. The top level domain `.no` is, for instance, a no-go.

Verifying an `RRSIG` RR signature To be able to authenticate a signature from an `RRSIG` RR, a resolver would need to follow the specifications in RFC4035 [2, section 5]. A short summary of how to authenticate a RR by validating the signature from its corresponding `RRSIG` RR:

- The target RR and its `RRSIG` RR must be valid, which means that the different settings, like the algorithm setting, must be understandable by the resolver. The TTL and the lifetime of the signature must be valid at the time of processing. The details in the `RRSIG` RR must match the target RR, for example must the registered owner name, class and type be the same.

- The `DNSKEY` RR that is referenced to by the `RRSIG` RR and which should be used to verify the signature must exist, be valid, and already be authenticated and trusted by the resolver.

  Since the *Key Tag* field is used for referencing to the public key, collisions might occur so more than one public key is referenced. In such cases, each public key should be used

- The data that was used for creating the signature must be gathered. This consists of all the data in the given `RRSIG` RR except for the

Signature field, appended with the set of RR that is signed, in sorted order [3, section 6]. The content of each RR is included, including the name, type, class, original TTL and the length of the RR.

- The gathered concatenation of the signed data gets sent through the secure hash algorithm and gets decrypted by the signature algorithm as defined in the `RRSIG` RR. An example on algorithms are the encryption algorithm DSA and the secure hash algorithm SHA-1. The referenced public key from its `DNSKEY` RR should be used when decrypting the signature.

  The output of the signature algorithm *must* match the signature as presented in the given `RRSIG` RR for the target RR to be authentic. If the result does not match with the given signature, the target RR is considered as invalid.

Authenticating the chain of trust For a resolver to authenticate an RR in a given zone, it must recursively verify each DNSSEC signature from the root zone and down to the given child zone, including the delegation signatures. The information could be requested from security-aware cache resolvers, or it could be requested from the authoritative name servers. See RFC4035 [2, section 5] for the details in how the authentication process should be performed by resolvers. For each zone it would process:

1. One of the parent zone's `DS` RR must reference to one of the given zone's `DNSKEY` RR. The parent zone's public key must already be authenticated or trusted, so we could verify that the signature in the `RRSIG` RR for the given `DS` RR is authentic.

   The `DS` RR authenticates the `DNSKEY` by a digest, which is created through a secure hash algorithm, e.g. SHA-1. If the secure hash output matches the targeted `DNSKEY` RR, the public key is considered authentic.

   If the current zone is the root zone, the resolver would need to have received the DNS root key through a secure extra-protocol channel and would not find a `DS` RR.

   If an `NSEC` or `NSEC3` RR is returned instead of a `DS` RR, and the signature in its corresponding `RRSIG` RR is valid, the given zone is not set up with DNSSEC and the authentication process stops here. It is up to local policy if the DNS data should be used when considered *insecure*.

2. A zone's public key is often only used as a KSK which signs the other public keys, the ZSK. The resolver would therefore have to authenticate the other `DNSKEY` RR by validating their corresponding `RRSIG` RR,

which must be signed by the KSK. In other words, the KSK signs the `RRSIG` RR for the other keys, while the ZSK signs the `RRSIG` RR for the rest of the data at the given name server.

3. The zone would either give the resolver the targeted RR, e.g. `TLSA` RR, or it could forward the resolver to the next zone in the route through a signed `NS`, `CNAME` or other RR types.

The route to the next zone is originally defined by the different parts of the host name, but Some DNS data could change the DNS route between the zones. For example would a signed `CNAME` RR require that the resolver asks a different name server for the information for the rest of the chain, and an `NS` RR would tell the resolver that another name server is authoritative for the requested information. The `DNSKEY` RR for all the name servers must be authenticated in the same way as for the rest of the chain.

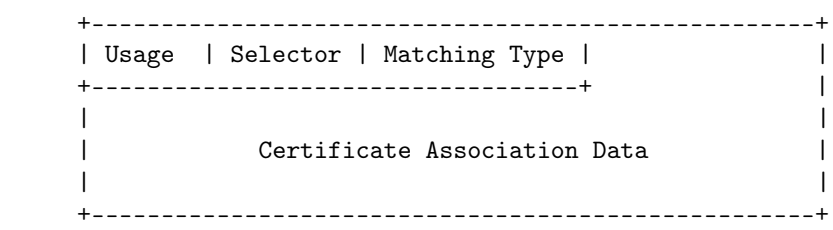
5.7 Using DNSSEC for authenticated data

As DNSSEC has been implemented and the DNS root key has been put into real life, DNSSEC has gained the trust of others to start using DNS for scenarios which requires authentication.

5.7.1 DNS-based Authentication of Named Entities (DANE)

The *DNS-based Authentication of Named Entities (DANE)*[10] is a *Working Group* under *IETF* that is developing standards for how to store public keys in DNSSEC in such a way that it could be used by clients as an alternative, or in addition to, the X.509 PKI. One of their results is RFC6698 «*The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*» [19], which is described further on in this section. This RFC could become the accepted standard in the future for using DNSSEC to authenticate hosts' certificates when using HTTPS, depending on if software and administrators makes use of the standard.

The Working Group have requested a new DNS Resource Record (RR) called *TLSA*, where keys and settings should be stored. The format of the TLSA resource record is:



[10] https://datatracker.ietf.org/wg/dane/charter/

The **Certificate Usage** part (1 byte) specifies how the *Certificate Association Data* should be used. There is defined four usage value types:

- **0 - CA constraint**: The certificate association data must match a CA's certificate in the certificate chain used in TLS.
- **1 - service certificate constraint**: The certificate association data must match the last certificate in the X.509 chain, i.e. the server's certificate. The PKIX chain must be validated as well.
- **2 - trust anchor assertion**: The certificate association data must match a trust anchor's certificate in the PKIX chain, even though it does not exist in the browser's CA certificate list. The service could with this define its own trust anchor.
- **3 - domain-issued certificate**: The certificate association data must match the server's certificate, and the X.509 chain should *not* be validated. This totally disables the PKIX validation, and tells the clients' to trust the DNSSEC chain instead, and the given certificate.

The **Selector** part (1 byte) specifies what part of the TLS certificate the association data should match, either all the parts of the certificate or its *SubjectPublicKeyInfo* [10], which means only the subject and its public key. The **Matching Type** part (1 byte) specifies what the association data contains, to be matched against the certificate, currently either the complete certificate, or a SHA-256 or SHA-512 hash of the selected part of the selected content. The **Certificate Association Data** contains the data that should match the certificate in the specified way, which means that it should either contain a full certificate or a hash of it.

The DNS URL for getting these RRs should be on the standard form

```
_PORT._tcp.domain
```

e.g. `_443._tcp.example.com` [19]. This is DNS' solution for being able to support using the RR type for different ports on the same host.

The point of the new DANE protocol is to standardize how certificates should be checked by clients when using TLS, especially web browsers. If a system has more specific needs, it could use other RR types for storing other certificates, but it would then need to have control of both server and client software, as you would need to implement special behaviour in both ends. With DANE'S RFCs you get the benefit of all clients following the same regime, which makes it more usable to adopt and implement for enterprises.

Downsides of TLSA

The TLSA specification looks, at first sight, complex since it requires DNSSEC to be used and it is built on top of TLS and tries to support

the variety of usage of TLS and the current PKIX validation. With regard to the various implementations of TLS it is mentioned that *local policy* might change the client behaviour of TLSA. The complexity and flexibility of TLSA might confuse administrators and decision makers.

TLSA has the option of totally disabling the PKIX authentication. This might sound dangerous to those who have full trust in PKIX today, as servers could then be set up without any CA involved. This might slow down the speed of adoption of the new standard, or it might even create client implementations of the standard that disregards the usage setting of ignoring PKIX and only authenticate through TLSA and DNSSEC. Client implementations ignoring parts of the standard might create difficulties in adoption of TLSA.

Another downside is that when using DNSSEC most of the requests would contain too much information for an UDP datagram, and the DNS resolvers would instead give its responses through TCP. Many network appliances does not handle this by default, especially in devices for home networking. [12]. This could make TLSA look unstable, as DNSSEC would not work in all places without modifications to the network equipment. Without DNSSEC working in all places, it would also hinder the adoption of TLSA.

As the PKIX is an active and widely used standard, even though it's complicated, the TLSA is new and tries to fix some of PKIX' vulnerabilities, it would take some time for it to be widely used. This is however not a downside if people already have mistrust in PKIX.

5.7.2 Storing certificates in DNS

There are, in addition to DANE'S specifications, other RR types for storing more generic certificates and keys in DNSSEC, which could also be used by clients. Such RR types are not limited to a certain usage, but servers and clients are free to use them however they like. If one has control of both the server and the client, it would be easy to use some of these to fulfill specific needs, with the downside of not using a common standard.

One of those RR types is the *CERT* RR specified in RFC4398 [28], which is for storing certificates. The type does not define how to use the certificates, only how it should be stored. This makes it possible to store a certificates in a standardized way, and clients that understands how it should be interpreted could make use of as it would like to.

Another RR type that could be used for certificates is the *TXT* RR. This is even more generic, as it could be used for anything both human readable and machine readable. As long as it is protected through DNSSEC it is usable for authentication, even though it is not standardized and could contain other data you would need to filter out.

A disadvantage of the *TXT* record is that the DNS zone owner might not

protect its values that much, and does not implement enough authentication credentials for updating it as one would with a certificate resource record. The *TXT* RR also lacks a specific format, which prevents the data getting validated before it could get put into DNS, which could then contain data not understandable for the systems that would like to use the data. If the format is partly bogus, or a *TXT* record looks like it contains a key, but is rather set up for a totally different system, a client could block the web site, and thus avoid any transactions from the site until DNS is updated.

5.7.3 Serializing DNS records with DNSSEC authentication data

The developers of the web browser Google Chrome and Chromium had implemented support for server authentication through the use of DNSSEC, [32] which they published an *Internet-Draft* for in [31].

Adam Langley, the main author of the Internet-Draft, employed by Google, thinks that the clients should not do the work as recursive resolvers, as that requires that the browser has to wait for many DNS requests before it could authenticate the service, which would take up time when waiting for a web site not already cached. Google wants as little as possible delay between user interaction and the display of a web site. It would also give less load on the DNS name servers, as the clients does not talk directly with the name servers, but get all the necessary DNS data from the server. [33]

Google's solution was used in Chromium for at least one year, but got later removed due to lack of use. The solution was to let the web server itself be set up with a self-signed X.509 certificate, but with a certificate extension where DNSSEC data is added in its raw format, containing every DNSSEC key and signature from the root node down to the web server, including the web server's *TXT* or *TLSA* RR, which should be set up with the service' public key. An example of a service that contains a certificate with DNSSEC data can be found at https://dnssec.imperialviolet.org.

The proposal was discussed in the DANE work group when the TLSA standard was developed[11]. A few downsides of the proposal was mentioned, where the main challenge was the added complexity of the system as a whole if the serialization should become a part of the standard, and the risk of problems when administrators that does not follow best practises start implementing it in their servers, leading to unpredicted behaviour and undocumented consequences. The proposal still has some advantages that is usable for servers that has their DNS settings set up correctly and follows the recommendations of the DNS and DNSSEC documents. One particular benefit is that the serialization goes around the problem of TCP data from DNS getting blocked, as mentioned in Section 5.7.1 on the preceding page.

[11]The start of the e-mail thread is found at http://www.ietf.org/mail-archive/web/dane/current/msg02758.html

When the given server certificate, received by the TLS handshake, contains an extension with DNSSEC resource records, the records could be authenticated by the browser, which has the DNS root key pre stored. The full path of the service must match the resource records, and the last RR, the `TXT` or `TLSA` RR, must match the given certificate from the service.

Google's solution creates a binding between the given service, the certificate and DNSSEC's chain of trust. The authentication process:

1. The browser and service perform a TLS handshake and the service responds with its service certificate containing the DNSSEC data.

2. The browser verifies the given DNSSEC chain of trust by its pre registered DNS root key. All the DNSSEC RR in the chain and their signatures has to be valid and must be verified for each zone. The host name of the service and its port number is used for finding the DNS path.

3. The browser verifies that the given `TLSA` RR contains the same public key as used in the certificate.

4. The browser accepts the self-signed certificate and continues with the TLS handshake with the service.

No CA is involved in neither the registration process nor the authentication process in Google's solution, as the certificate is self-signed. The registration and authentication processes has instead been dependent on DNSSEC, which prevents certain network attacks.

Only the service' administrator should be able to modify the `TXT` and `TLSA` RR for the given service, which means that attackers have to either attack the given administrator or the registrar of any of the zones from the root node to the service' zone. In contrast, when a service is using a certificate from a CA, an attacker would, as previously mentioned, have to attack either *any* CA, the registrar of any of the zones from the service to the root node, or it could exploit vulnerabilities in the PKIX.

Due to the lifetime of the DNS RR, service certificates with DNSSEC extensions needs to be periodically regenerated after a few days, since modified extensions affects the signature of the certificate. The service must have a routine to periodically generate and make use of an updated certificate in its TLS server.

The certificate with the DNSSEC extension is following the X.509 standard, with the DNSSEC data in its own extension. The DNSSEC data is formatted as *c struct* data[12], which makes it easy to parse by clients[13].

[12]The code for both generating the extension and the certificate is located at https://github.com/agl/DNSSEC-tls-tools

[13]Chromium's code for parsing the DNSSEC data is located at http://src.chromium.org/svn/trunk/src/net/base/dnssec_chain_verifier.cc

The DNS dump contains the target RR which is a `TXT` or `TLSA` RR, followed by the DNSSEC data for authenticating the RR, from the service' zone to the root node. For each zone it contains:

- Every `DNSKEY` RR.
- The name of the zone, for easier lookup.
- The `CNAME` RR, if defined for a node.
- The `DS` RR to the next zone to verify the next `KSK`.
- The `RRSIG` for every RR, except for the `DNSKEY` which is signed by the parent zone's `DS` RR.

The solution expects all DNS data to exist. All `NSEC` RRs, for missing data, are ignored by the solution.

The benefits of letting the server generate the DNSSEC authentication data:

- The service administrators would not have to buy a signed certificate from a CA, but could instead rely on DNSSEC.
- The browser would save time in that all necessary authentication data is received from one source, in one request.
- The DNS infrastructure would get less load, as only the given service will request the given information, and only once per day or less, depending on the lifetime of each RR.
- Browsers in locations would still get access to a service which authenticates through DNSSEC, even if the local routers is set up badly and blocks DNS that goes over TCP and not UDP.

The downside of server generated certificates with DNSSEC data:

- The server has to be updated to make use of the new certificates.
- The service gets an extra routine, as the new certificate has to be regenerated regularly, and the service must make use of the new certificate at every regeneration.
- The solution is not backwards compatible. Existing browsers not supporting Google's solution will not be able to authenticate such services, and the user will get a warning.
- There is no fall back solution if the service' zone does not support DNSSEC. Without DNSSEC you would not be able to create the certificate, and would instead have to create a CA signed certificate. For the solution to work both with a CA signature and DNSSEC the CA would have to support an automatic and daily resigning of the certificate.

Chapter 6

Summary

The theory part of this thesis has presented basic functionality that is security related, with a focus on what is needed for service authentication. A general introduction to the basics of authentication, security usability and public key infrastructures was also needed for the later discussions.

When it comes to PKI systems, the theory shows that the current PKIX standard is considered to be outdated due to its vulnerabilities in how it is designed and how it is in use. Other PKI standards have been introduced, and the standard named *TLSA* is one of the candidates for either replacing the old PKIX, or living side by side of it, when it comes to authenticating online entities. TLSA has therefore received more focus than other PKI systems in the theory part. The TLSA standard is itself smaller and less complex than PKIX, and it is dependent on DNSSEC's chain of trust. It could still be dependent on PKIX, depending on its service settings. Is TLSA usable, and beneficial, for an offline device such as the OffPAD?

The principles behind *Security Usability* tell us that it is important for the end user to understand the process, and the importance of giving the end user information that is understandable. The user might not have its full, mental capacity in the heat of the moment, e.g. when in a shop. How could the usability be taken into account if the OffPAD should authenticate services through DNS, a service which for most people seems rather complex?

The TLS standard has been covered in principle. We also understand that the TLS standard is used differently in different environments, both because it only specifies the protocol and not how the protocol and certificates should be used and given, but also because different TLS implementations have changed its behaviour for corner cases and by extensions of the X.509 certificates. Does the OffPAD need to consider these differences in TLS and X.509 implementations for its service authentication?

DNS and DNSSEC have been covered in basic, as it is crucial to how the OffPAD should do the authentication when not using PKIX. This is a more complex standard as it builds on various RFC documents and has been

updated and superseded by later RFC documents. Will support of DNS and DNSSEC make the OffPAD too complex to implement?

One thing the theory part has not covered directly, is how service authentication would be solved when in an offline environment. The DNS standard requires you to have access to a name server from time to time, as the information from DNS has a limited life time. How would this be implemented for devices such as the OffPAD, which is offline most of the time, and is not able to communicate with name servers directly? The theory has covered Google Chromes' old solution, where the DNSSEC chain is serialized and wrapped inside the certificate extension as given by TLS. Could Google Chrome's solution be usable in the offline environment of the OffPAD as well?

Offline devices have a few challenges that the theory has covered, but not given a solution to. How could the device be able to determine if an encrypted HTTPS stream is the correct stream for the correct site? As history shows with PKIX, where only the host name is validated and not its IP address, we need to make sure that the OffPAD have no gaps in the validation that could be exploited. Offline devices needs to authenticate the messages that it receives, but since they receive the information through other entities, how could the device make sure that the information is authentic? Should TLS sessions be authenticated? Must the service sign its messages differently than today for the OffPAD to fully authenticate a service and its transactions, or could the OffPAD still be used for existing services without modifying their behaviour?

DNSSEC has some new challenges that were described in the previous chapter. The first challenge is the time it takes for a client to retrieve all the needed data from DNS to validate the full chain of trust. For impatient end users of the web even 250 milliseconds could be long enough for them to change to a competitors web site [37]. This makes every millisecond important, and having to send a few more DNS request might be too much for the industry to accept that each entity should authenticate the chain of DNSSEC. Another issue is that DNS requests through TLS gets blocked in various equipment, especially for private use [12]. How could we avoid that these issues affects the OffPAD?

Part II

The project

Chapter 7

Method

This paper is a part of the *Lucidman project*[1], which is both designing a theoretical OffPAD and creating a prototype of it. Many details in this project is therefore dependent on other parts of the Lucidman project, as the overall specification makes the foundation of this thesis' proposal. All the use cases related to service authentication is mentioned to get an overview of what is required by the proposal.

In my part of the project I must get to know the theory about PKI, PKIX, DNS, DNSSEC, TLSA and how to use it properly and securely. I will then create my proposal on how this could be implemented in an OffPAD for the authentication to be secure. I start with describing the details of how service authentication with PKIX would be implemented, to be able to understand and see the differences with how the service authentication process with TLSA should be implemented.

The solution of thesis is done through theoretical research, and testing the validity in theory. Theoretical scenarios are used to test the solution.

[1] http://lucidman.org

Chapter 8

The Lucidman project

The Lucidman project is a research project on security-usability, where the focus is strengthening the security for the end user. It is a cooperation of different organizations located in both France and Norway:

- University of Oslo, Norway, which are working on the security-usability.
- GreyC, an academic research laboratory that is part of ENSICAEN, a technology school which are doing research together with industrial partners. It is located in Caen, France. Their focus is the identity management in the project.
- Tellu AS, a Norwegian company that is working on applications for the mobile domain. Their focus is mobile applications in the project.
- Vallvi AS, a Norwegian consulting company working on commercialization of new technologies. Their focus is the business development of the project.
- TazTag is a company in France that develops secure mobile appliances. Their focus is the hardware needed in the project.
- CEV (Chèque Déjeuner Group) is a company from Normandy, France, which develops software solutions for loyalty smart cards. Their focus is the business cases.

The project's web site is located at lucidman.org.

The motivation of the Lucidman project is the problem of *password fatigue* and *identity overload*. End users need to generate a user name and a password for each website they are using. This goes all well if they are only using a few websites, but end users are today working with more websites than they are able to remember the passwords for. The end user solves this by workarounds, which could either be writing down the passwords or they could reuse a password for different services. Today's password regime has a

bad security-usability, making it hard for the user to work securely and easy to make mistakes. A user could easily lose the paper or data file where the passwords were written down, and by using the same password everywhere you have the risk of getting compromised everywhere if you lose your one password.

Another security-usability issue for the end users which the Lucidman project is working on, is the *cognitive server authentication*. The end user must be able to authenticate a service before it should be used. Phishing is an example of attacks that works effectively today due to various reasons, one of them being the difficulty for users to tell the difference between for instance http://facebook.com and http://faceboourk.com[1]. Checking a URL is especially difficult when the URL contains a lot of other data, or it is wrapped inside a URL-shortener, e.g. http://bit.ly. The end users does not have much tools to work with to be able to figure out if a service is authentic, and it is too easy to send the credentials in the wrong direction.

The last security-usability issue, that affects the previous ones, are the problem of infected computers. The end user's computer could be infected by a trojan that either captures credentials when authenticating for a website, or it could be faking a real website the user would be using and sending the credentials back to the attacker. Now the user has even fewer possibilities to secure its own identities.

A challenge for changing the current way of authenticating users online today, is that most solutions are costly for the enterprises that runs the services, which is why we are still talking about passwords, even though there exists more secure alternatives. New authentication solutions should therefore take in consideration the costs of the enterprise, before the solution is usable.

The project's solution to solve many of the security-usability challenges is for the end user to have a device which is dedicated to manage the user's identities and to give the user tools for authenticating services. The device should mostly be offline, and should only be communicating with other devices, like the user's computer, when asked, thus avoiding getting compromised. This device is in the project name the *OffPAD*.

8.1 The OffPAD

The *Offline Personal Authentication Device (OffPAD)* is a device that is focusing on security-usability, and should give the end user tools for securing his' or hers online identities to prevent man-in-the-middle and phishing attacks while managing the user's online identities. The initial proposal of the *Personal Authentication Device (PAD)* was published in 2005 by A.

[1]Which was the URL of a real phishing site in January 2013 - http://www.troyhunt.com/2013/01/please-login-to-your-facebook-account.html

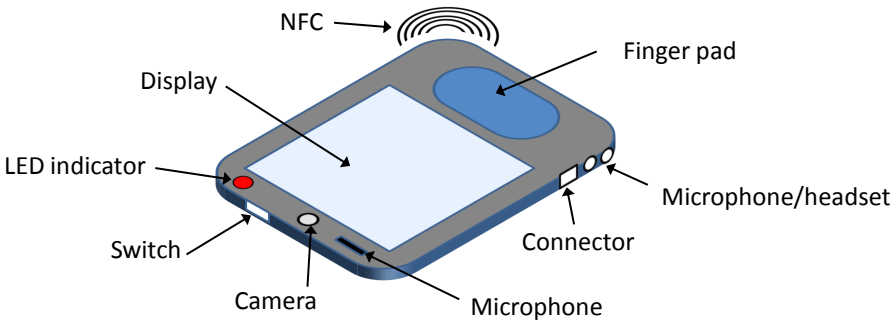


Figure 8.1: The OffPAD and its physical features [23]

Jøsang and S. Pope. [25]. An improved version of the PAD, the OffPAD, was proposed by Klevjer et al. in 2013. [29]. Another device, the Nebuchadnezzar [35] is a device that matches the idea and specifications of the OffPAD.

The OffPAD has the roles of being:

- An *identity management system*, keeping control of the end user's credentials for various services. This includes being a *password management system*.
- A *Service authenticator*, helping authenticate services the user should communicate with.
- A *transaction signer*, helping the user sign data electronically.

An example of how an OffPAD could look like is shown in Figure 8.1. The OffPAD should be small enough to be easy to carry, it should have a screen to inform the user about what is happening and input methods to get feedback and authentication attributes from the user. By using such a device, the end user would not need to remember the password for each service, and the services would not need to change their infrastructure, as the OffPAD could store the existing user names and passwords. The device would be able to support other authentication methods for those services supporting it, e.g. challenge responses and public key authentication.

One part of the OffPAD's tasks is checking the identity of the service it should be handing out the credentials for. To avoid man-in-the-middle (MITM) attacks, it must make sure that the service is the correct one, so it must be authenticated somehow. As is already mentioned in the theory part of the thesis, we already know that the current PKIX regime contains flaws, we want the OffPAD to be able to authenticate the services through DNSSEC as well, for the services which are using this. This is the part of the Lucidman project where this thesis comes in.

8.1.1 Requirements

An example of some the physical features of the OffPAD is shown in Figure 8.1. Some of the OffPAD's requirements from [44, 29] are:

- The device should not be of general purposed usage, to limit its attack surface. The device is meant to be security focused.
- Various cryptographic functions must be supported by the device.
- The device must support updates, as it must be able to be updated with security related fixes. The update must be done in an equally secure way as the authentication of other transactions, to avoid that the updates becomes an attack surface.
- The device should have limited connectivity to limit its attack surface, both in time and in range. The device must be offline until the user tells it to get online, and only for a short period. By using short range communication methods, like NFC, the communication gets limited in distance.
- The device must have access control. The device must authenticate the end user with a PIN, password, pass phrase, biometrics or a combination of various attributes. The user must be authenticated for each session, which is limited to time or connection.
- Physical tamper resistance is a requirement. An attacker with physical access to the device should not be able to access or modify information stored in the device, or modify the parameters of the device. Data must be encrypted on the device, and it should only be able to decrypt by the user's feedback.
- The device must support transient, replay protected, challenge-response communication with service that supports this. The communication goes through an untrusted third party, typically a user's computer that might be infected with viruses. If the service does not support challenge-responses, it should fall back to regular authentication with username and password.
- Each service and transaction must be authenticated before transactions could be processed. The device C must authenticate the server S, but *cognitive server authentication* is also required. This prevents both some kinds of phishing attacks and network attacks. The device' authentication process must involve DNSSEC, which is the focus in the proposal part of this thesis in Section 9 on page 63.

By combining these requirements, you would get a device that could be trusted for your credentials more than software on your general purposed computer or smart phone. One assumption is that you trust the vendor that has produced the device.

8.2 Use cases for the OffPAD

The generic features of the OffPAD makes it usable in various scenarios and environments, and it could support different use cases. The various use cases have different security focus and requirements. The OffPAD could both be used together with the user's computer, which is an environment the user is familiar with, but it is also functional when out shopping, in which stressful situations could occur, joined with an unstable or slow connectivity. This section contains the use cases for the OffPAD in which server authentication is required, and where the security requirements are different.

8.2.1 Managing credentials for online services

One of the main use cases for the OffPAD is the scenario where the user wants to authenticate herself to a web service, and lets the OffPAD handle this by first authenticating the service and then handing out the credentials to the service. For this to work, the OffPAD needs to store the credentials for the services in an encrypted format, and the OffPAD and the computer needs to be able to communicate.

The typical scenario for handing out credentials:

1. The end user is located at an untrusted computer. The computer could either be the user's own, or it could be a random computer, but it can not be trusted. All computers could in theory be controlled by an attacker, so you would for instance expect trojans.

2. The end user is going to a service which wants to authenticate the user.

3. The user recognises that the service asks for credentials. If the service supports more secure authentication protocols, like the *extended HTTP Digest Access Authentication* protocol, the computer might be able to recognise that the service requires user authentication by itself and would both automate the process and avoid that the credentials get sent in plaintext.

4. The user activates the OffPAD by authenticating himself for it.

5. The OffPAD authenticates the service, e.g. through use of DNSSEC.

6. The OffPAD inform the user if it has already made use of this service before or not, e.g. by using the *Pet Name system* [15].

7. If the OffPAD has stored credentials for the service, the user is asked if this should be given to the service. The OffPAD would *only* give the user the option to hand out credentials for the correct service URL. All other service addresses would not be recognized, as that could be a phishing attack.

If the user is going to a new service or a service which the OffPAD has not stored any credentials for, the user would need to be able to register the service and its credentials to the OffPAD for later use. First, the service should be identified and registered using the *Pet Name* system [15], so the user would recognise it the next time. Second, the credentials should be registered for the given service, which could be old fashioned username and password, one-time passwords (OTP) or a challenge response protocol. When regular forms with username and passwords are used for user authentication, the OffPAD might not know exact what should be stored, even though software might help the user in knowing what should be stored, e.g. by reading the identifiers of the credential form.

8. If the user accepts the service, the credentials are transmitted to the computer, which should pass them on to the service. If the credentials consists of a regular username and a password, the computer would be able to see them in plaintext, which is not the case if the more secure *Extended HTTP Digest Access Authentication* protocol [29] is implemented at the service.

9. The OffPAD goes back to sleep.

10. The user gets authenticated by the service.

The OffPAD is only involved when the service should get authenticated and the credentials should be stored or given. The rest of the transactions with the service is handled by the user and the computer.

Types of credentials

The OffPAD could support more types of credentials than an end user might be able to make use of by hand:

- Username and passwords that might be longer and more complex than a user would be able to remember.
- Challenge-response protocols.
- One Time Passwords (OTP).
- Public key authentication.
- Hash-based Message Authentication Code (HMAC)

Attack scenarios

The registration step is an important step regarding the security usability, as the user must now be able to understand if this is the correct service to authenticate for. The service must be authenticated, e.g. through DNSSEC, but since it is the initial authentication process, the user is still vulnerable to phishing attacks. The registration process is often a difficult situation in terms of authentication, as you often have no previously stored information to compare with. By using the OffPAD in the registration process, you are at least not vulnerable to today's network attacks, as the service authentication checks that the service is for the correct address.

Should the user be able to register credentials for a service before actually going to the service? This could speed up the process the first time the credentials are actually needed, but at the same time it also opens up for phishing attacks. To be able to store credentials for a service, you need to be able to add its full URL, which could be a different address than what you thought of, if someone was giving it to you, for instance through a fake e-mail.

It should not be possible to edit a URL for already stored credentials without visiting the service by its new URL. Changing a URL could be a phishing attack scenario, and must be taken care of carefully.

If the user has previously registered for a service, and the attacker tries to pretend to be that service, it could tell on the front page of the service that all the credentials must be reregistered for the OffPAD. When the user tries to give the service its credentials, the OffPAD would not recognise the service, since it is located at a different URL. If the user believes the message about the technical difficulties, it would try to reregister the service and either feed the OffPAD with its credentials for the wrong URL, or manually give the credentials to the fake service.

The registration and identification processes on the OffPAD must be implemented in a way that gives the least form of confusion and complexity for the user, and give as little room as possible for errors that could lead to phishing attacks. As with most new technology, the regular user will not be able to understand it enough, which is something phishing attacks are exploiting. The use of an OffPAD is a new process for the user, so the new processes it gives must be as understandably as possible.

8.2.2 Managing credentials for physical locations

A use case that looks a bit similar to managing credentials for online devices, is the management of credentials for physical locations. At some physical locations you would like to be authenticated, for instance at a security entrance at an office, airport or a post office for handing out your registered packages. There are different levels of security requirements for

access control. A requirement for making use of the OffPAD, is that the location has an authentication device, here called the *service*, that could communicate with the OffPAD, e.g. through *Near Field Communication (NFC)*, and through an authentication protocol available to the OffPAD.

There are advantages for using an OffPAD for physical access control:

> A fingerprint can be scanned and validated on the OffPAD instead of on a central system, which enhances convenience, hygiene and privacy for the user. After matching the fingerprint the OffPAD can send an assertion to the access control system. [44]

Such an assertion must contain enough information for the access control system to authenticate the user and to avoid replay attacks, while at the same time not be reversible to protect the privacy of the user.

The scenario could look like:

1. The user authenticates herself to the OffPAD and activates it.
2. The location's service must send a request to the OffPAD for its credentials. The request could contain:
   - A challenge that could be solved with the user's credentials. This could for instance be solved using data from the user's fingerprint.
   - The time or a *NONCE* to avoid replay attacks.
   - Authentication data for the authentication device, normally the public key or a certificate with the public key.
   - A chain of certificates that has signed the authentication device' public key. This could be used by the OffPAD for service authentication.
   - The request must be signed by the public key of the location's device.
3. The OffPAD would authenticate the device that made the request.
   - If the device would *not* get authenticated, the end user should be notified of this. The user should not be able to override this and still send the credentials to the device. The user would for instance not know if the authentication device is tampered with, and stressful situations could lead to the user just wanting to get through the authentication process. If such a situation occur, it should be handled in other ways than forcing out the credentials.
4. The OffPAD authenticated the device, and returns a response to the given challenge.

5. The service tries to authenticate the user by the given credentials. It must trust that only the user would be able to make use of its own OffPAD.

Attack scenarios

An attack scenario for devices with NFC, is that an attack device could get close enough and try to steal the credentials. Since the OffPAD is offline until the user activates it, you would not be able to steal the information without active help from the user.

Another attack scenario is faking the service, by swapping the location's authentication device by a fake one that could record the given credentials. The user could get tricked into trying to send its credentials to the fake service without knowing it. It is important that the authentication request is signed, so that the OffPAD could authenticate the request. As long as the fake service does not know the private key of the authentic service, the OffPAD would not send its credentials, and the user's privacy is safe.

Eavesdropping could be used to read out the credentials for a user, unless a protocol like the extended HTTP digest access authentication protocol is used. If confidentiality is required, it could be prevented e.g. by encrypting the challenge response by the service' public key, so that only the service would be able to read it.

First time registration of credentials for a physical location might be a situation that is open for attacks, depending on how the registration process would occur. In the registration process we might not have a way to cognitively authenticate the service, and we might not have any certificate or public key that could be used to identify and authenticate the given public key.

8.2.3 Loyalty cards

In some countries it is common to have *loyalty cards* for shopping. By using a loyalty card you, as the buyer, are able to get better prices, and you are motivated to keep shopping in the given store. The benefit for the store is to keep its customers.

By using a loyalty card, you would be able to register what you buy on the card, in which you could turn into offerings and better pricing. For the loyalty card regime to work, the shop would need to identify you to register how much you have bought for, and you would want to be authenticated to avoid that others could get the offers you have saved up for.

The motivations for the OffPAD to store loyalty cards are almost the same as the use case for the OffPAD to *managing credentials for physical locations*.

8.2.4 Signing online transactions

When the user is online and needs to do a transaction, you would like the transaction to be authenticated by a digital signature from the user, even if the user is already authenticated to the service where the transaction is processed. A signature could be used for integrity and message authentication, and not only user authentication. This is for instance required for setting up contracts for buying houses, getting a loan and other transactions involving some amount of money.

Normally, a bank transfer today is done in a web browser and the user confirms the transaction e.g. by a one time password. When a signature is required, the user is sometimes even required to sign papers by hand, as there are yet no suitable alternatives required by law.

When a transaction is only viewed and accepted inside the web browser, it has no integrity and is open to MITM attacks, in which the information displayed in the browser might not be the actual data. An attacker could display the data from the user's previous input, but in the background modify it for its own benefit.

By making the OffPAD do the signature of online transactions, the OffPAD would support integrity in that only the information shown on the screen of the OffPAD gets signed. If the OffPAD is given information from the attacker, it would be shown in clear, and if the attacker tries to modify the information after handed back from the OffPAD the signature would fail. The OffPAD could be set up with a private key, and the bank could have registered what public keys that could be used to confirm the transactions for the user. More details about this functionality is found in [44].

The typical use case for signatures would then be:

1. The user authenticates with the online bank system. This could be done by the OffPAD, as described in Section 8.2.1 on page 56.

2. The user either sets up, or gets, a money transfer in the browser.

3. Before the bank processes the transaction, it asks the user to sign it. If the bank service supports it, the data could be transmitted to the OffPAD automatically.

4. The end user authenticates himself for the OffPAD.

5. The OffPAD

The benefit of having the OffPAD sign all the transactions, is that a virus would not be able to modify the parameters, e.g. the account the money should be sent to, without it being displayed to the user on the OffPAD's screen.

As the user could be confused in confirming a transaction with some attributes modified without noticing it, the use case could be changed to make more of the transaction be processed on the OffPAD:

1. The user authenticates with the online bank system. This could be done by the OffPAD, as described in Section 8.2.1 on page 56.
2. The user sets up a money transfer *on the OffPAD*. Now an infected computer would not be able to fake any of the data.
3. The OffPAD signs the transaction and sends it to the bank service.
4. The bank system recognises the signature and accepts the transaction.

Note that this use case requires that the service is able to send the data about the transaction in a standardised way, for it to be usable by the OffPAD. Also, in the last version of the use case, the OffPAD is required to have software that supports creating a bank transaction on the OffPAD, with the downside of not being that flexible and modifiable by the bank itself.

The signature must contain enough information to avoid replay attacks.

8.2.5 Encryption and decryption of messages

Since the OffPAD both has a secure storage for private keys, and supports security algorithms, it would support authenticating and singing messages. It is then also usable for encrypting and decrypting messages by given keys, which could for instance be used to transfer confidential information.

PGP is an example of a PKI which also supports encryption. Since we can't trust the computer due to the risk of getting compromised, the OffPAD is a suitable place for handling both the public and private keys, and the encryption itself.

Chapter 9

Server and Service Authentication with the OffPAD

This chapter describes how the OffPAD can be used to authenticate services for the end user to prevent man-in-the-middle (MITM) attacks. The proposed service authentication is supposed to work for most of the use cases of the OffPAD, and any difficulties are addressed. The OffPAD must be able to fully cover the authentication modality of the *cognitive server authentication*, $\mathbf{S} \rightarrow \mathbf{U}$, as mentioned in Section 5.3.1 on page 19. The proposal covers the syntactic server authentication of $\mathbf{S} \rightarrow \mathbf{C}$ (server authenticated by the client) in details, but includes information about the cognitive modality where necessary, as the proposal is worthless if the cognitive server authentication is not implemented. More specific details about the cognitive server authentication functionality is covered by other these and documents, for instance by the *Pet Name system* [15].

9.1 Use cases for service authentication

The OffPAD needs to authenticate all services for all its use cases. Some use cases would benefit from using DNSSEC, while for other use cases it might be considered, but for the most use cases the service *must* be authenticated before it could be used by the OffPAD and the user. Service authentication is especially important when protected information should be given away, for instance client credentials for a service.

The main focus of the proposal is how the service authentication could make use of credentials stored in DNS, and how DNSSEC could be validated in the offline device. The various use cases of the OffPAD have different requirements regarding the service authentication, and not all use cases might be usable for authentication through DNS. This section discusses the

various use cases with a focus on how they could work with DNSSEC for service authentication.

9.1.1 Storing credentials for online services

The use case of storing credentials for online services, as described in Section 8.2.1 on page 56, is one of the primary goals of the OffPAD, and is also the use case that looks most suitable for service authentication by using DNS, as there is a strong connection between a service, its IP addresses, host name and DNS data, and being online is a requirement to be able to use the service.

When the user wants assurance about the server identity, e.g. before entering its credentials, the OffPAD must first authenticate the server's certificate, host name and other attributes, as given by the online device, e.g. a computer. The certificate must be validated by checking the signatures in DNS, which must also be given by the online device, and the chain of signatures in DNSSEC must be validated before the service could get authenticated. When the server is syntactically authenticated, the user must be included in the authentication process and asked for confirmation about handing out the credentials.

Online services could be used at the user's own computer, but the user might also make use of other computers when communicating with a service, for example when travelling and using a hotel's computer available for all guests. In *any* situation, the computer through which the user is communicating should be trusted at least as possible, as it might be infected with Trojans or being under surveillance by an attacker. The challenge for the OffPAD is that it should not trust the data from the computer, as it is a potential man in the middle, while at the same time being dependent on the client to be able to communicate with the service.

The risk of breaking or circumventing the service authentication is that the credentials for *any service* the OffPAD is storing could get leaked to the attacker. If the service authentication is broken, even the *HTTP Digest* protocol is vulnerable to MITM attacks, as the computer could give the OffPAD its own, fake challenge that could be easy to reverse, attempting to be some other service. The challenge itself does not contain a service's signature, which makes this possible.

The OffPAD must expect that all the information coming from the computer could be faked, for the computer to be able to steal the credentials. Every part of the data communication should be authenticated, and the OffPAD must make sure that the given data is actually sent from the service. How could all the data from the service be authenticated by the OffPAD before handing out the credentials? We do not want a missing link as in X.509, as mentioned in Chapter 4 on page 11.

Attacks on the OffPAD are, however, still protected by user interaction.

The user must be involved in the service authentication, and must confirm each credential request before it is given to a service, and the user interaction must follow the principles of security usability as mentioned in Section 5.2 on page 16. If the service authentication is implemented in a way that is usable and understandable for the user, we are protected against today's phishing attacks and other attacks, as the user would be able to discover that something is wrong.

9.1.2 Storing credentials for physical locations

The use case of *access control*, as described in Section 8.2.2 on page 58, is a bit similar to the online usage, but with other challenges, as a physical location is often thought of as offline.

If the physical location would make use of DNSSEC for service authentication, it would require that the location's authentication devices all had their own identifiable address, either by a host name, an IP address or an identifier that could be registered, and found, in DNS. IPv6 would make it suitable for each device to have its own IP address. A certificate for each authentication device, or a shared certificate for all the devices of the location, could be stored in DNS. The access control devices does not need to be reachable online, as the OffPAD has a direct communication with the device.

Advantages of validating access control services through DNS:

- The administrators of the access control system could change the certificate in DNS, and the OffPAD would accept it, as the chain of trust in DNSSEC would authenticate the new certificates.
- Revocation of certificates would be supported. An authentication device could get stolen and misused, and if the OffPAD did not do an online authentication of the certificate it would not know that the authentication device was not in use anymore.
- The initial authentication process would still be open to phishing attacks, but we could at least authenticate the device by DNSSEC's chain of trust.
- If the *Pet Name system* would be implemented for the access control, the IP addresses could be a placeholder for identifying the location's device.

The disadvantages for using DNSSEC for authenticating the authentication devices:

- The vulnerability of bad connectivity. If the access control would get offline, the OffPADs would not be able to authenticate the authentication service and would block the access control.

Due to the caching in DNS, the process could still work for a few hours if the location were caching the DNS data which would be needed locally, which would decrease the severity of being offline.

The risk of bad connectivity should be compared against the advantages of being online, like the support for revocation. Many locations are dependent on being online in any cases, for instance due to *Voice over IP telephony (VoIP)* functionality.

- The speed of the authentication. In the simplest form of service authentication without DNS the OffPAD would only need to validate a given signature by a pre stored public key. When DNSSEC is involved, the device must verify all the signatures in the chain of trust in DNSSEC in addition to authenticate the last signature. The data transfer comes in addition.

  In settings were speed is required, like in airport security, it might be a problem if the process would take a few seconds longer per passenger.

- The authentication device must support feeding the OffPAD with DNS data, which would complicate the requirements of the device.

The physical location *could* make use of DNSSEC for service authentication, but it might not always be beneficial, depending on the type of access control.

9.1.3 Loyalty cards

The use case of *loyalty cards*, as described in Section 8.2.3 on page 60, is mostly similar to the use case of access control, but in many cases less demands for security.

As in the use case for access control, DNSSEC could be used for service authentication, but it would require the store's device to support this. By using DNSSEC the OffPAD would support revocation, but it would at the same time cost more in requirements of the stores' equipment, connectivity and each use of a loyalty card would take longer time to process. If the amount of money involved in the loyalty cards are below the total amount of resources required to support DNSSEC, supporting DNSSEC would not be profitable for loyalty cards.

9.1.4 Signing online transaction

The use case of *signing online transactions*, as described in Section 8.2.4 on page 61, is usable for DNS in the same way as the use case for managing credentials for online services.

Even though DNSSEC would strengthen the service authentication, it might not be that crucial for this use case due to what information is

transferred and how the initial keys are registered. All transactions are signed by a private key in which the bank must verify. If this information should go to a fake site, it would not be able to do anything more with it than reading it, thus breaking the privacy of the user. The signature prevents it from being modified, and the signature should contain information to avoid replay attacks. DNSSEC would be implemented for protecting the authentication credentials for the bank service, and to protect the privacy.

For the procedure of signing transactions to work for the OffPAD, the device' public key must be given to the service the signatures should be given to, and the service' public key must be given to the OffPAD. The key exchange between the service and the OffPAD requires service authentication. The easiest procedure for getting the service' public key would be to store this key in DNS directly, without having to communicate through HTTPS. By storing the service' public key in DNS the OffPAD would be able to authenticate the key directly. The device' public key would be given to the service through regular user authentication, which could make use of the credentials stored in the OffPAD. The device' public key could be encrypted with the service' public key at delivery. It is recommended that the OffPAD generates its own key pair, and not the service, to avoid that the private key could get compromised.

9.1.5 Encryption and decryption of messages

The use case of *encryption and decryption of messages* is described in Section 8.2.5 on page 62.

The management of the public/private keys is the same as in the previous use case - *Signing online transaction*. If messages should be encrypted and decrypted for services, we need to authenticate the service at the time of registering its public key. When the public key is first registered, only the cognitive service authentication is needed. Data encrypted with a service' public key would only be readable by the service itself, so in this use case we avoid some confidentiality challenges as is mentioned in the Credential use case and in Section 8.2.1 on page 56.

If a message should get encrypted or decrypted for another service, you would need the public key of the service. DNS is a suitable place for storing public keys for services, as it is efficient and we only need to authenticate the DNSSEC chain of trust for authenticating the public key.

If a message should get encrypted or decrypted for another user, the other user and its public key must be authenticated. DNS has some support for this today already, in that each user could have their own data registered in a zone in DNS, for instance under their company's zone. Each user in a company could for instance get their own public key put under `USERNAME.users.company.example.com` in for instance a `CERT` RR.

For services to be able to receive the OffPAD's public key, regular user

credentials could be given through regular user authentication as today, but the user's public key could be encrypted by the service' public key to make sure that it is given to the correct service.

There are different protocols for encrypting messages. In TLS, for instance, the public/private keys are only used initially to establish a session key, which is used for further communication which goes through a symmetric encryption stream. PGP, as an other example, is encrypting messages by the public/private key directly. The OffPAD should support the different protocols that is required in the use case.

9.1.6 Getting authenticated data

This is not a specific use case, but rather a feature that could be used for different use cases. Since we are already using DNSSEC, and DNS could be used to store more than just service certificates, the OffPAD could be given data from DNS, which it then could authenticate. The resource record `TXT` could for instance contain arbitrary data, which could be used for anything.

Examples of use cases for this feature is the use case for *Encryption and decryption of messages* for getting users' public keys, and for *Signing online transactions*, for getting the bank's public key.

The information from DNS could be used in arbitrary ways, but there is challenge in linking the data together with the correct service, since the OffPAD is dependent on the computer for online communication. We must avoid that the computer could trick the OffPAD and the end user into believing that some DNS data is for a different service than what it actually is.

9.2 Authenticating services through PKIX

The theory part of the thesis has mentioned various authentication protocols, but the focus has been on the widely used PKIX and the new TLSA. Since many usage types of the TLSA standard depends on the PKIX authentication, and since TLSA is not implemented widely yet, and it would probably take some time before it gets popular, is the OffPAD required to support PKIX authentication. Without PKIX authentication would the OffPAD have limited usage for managing credentials.

The PKIX regime is described in Section 5.4.4 on page 30, and the service authentication for web browsers normally happens through the TLS protocol, as described in Section 5.5 on page 34. The authentication happens by the client checking the service's given certificate and the chain of issuer certificates all the way up to a CA certificate, stored locally. If all the certificate's signatures are verified and the timestamps are valid, the service is considered authenticated and the communication with the server could continue.

For the OffPAD to be able to authenticate services through PKIX it must have an internal list of its trusted CA certificates. When asked for authentication of a service, the OffPAD must be given the service' X.509 certificate, and any other certificate that lies between the service and a trusted CA. The PKIX validation must follow its specification, as each certificate in the chain must be validated. If the authentication gets accepted, the OffPAD would give the computer what was requested.

9.2.1 Attacks on PKIX

The attacks on PKIX, as mentioned in Section 5.4.4 on page 31, affects computers talking directly with a server, but the same attacks will also affect the OffPAD.

- The complexity of the PKI structure and the amount of CA certificates makes it easier to attack PKIX. The list of CA certificates put in the OffPAD must be thought through, and must be as short as possible to lower the risk of attacks. The challenge is that if the list gets too short you are in risk of blocking access to some, valid services.

- No CA is in PKIX technically limited to what domains it could sign certificates for. This is by the specification of PKIX, and it is not likely something the OffPAD could protect against. There is an extension to the X.509 certificates that implements name constraints for issuers, but the OffPAD could not make use of this as long as the industry does not use it.

- In the PKIX validation, there is no connection between the service and the registered IP address for the given host name, which opens up for network attacks. The network attack happens at the computer, before the information has reached the OffPAD, and the OffPAD does not have any more information to confirm or deny if such an attack occurs, which makes the OffPAD still as vulnerable to this attack as the computer.

  The same kind of network attack might also be implemented between the computer and the OffPAD if there is no connection between the given service' certificate and the data for consecutive transactions. An example of this is the use case of handing out credentials for a service. An attacker could first give the OffPAD the authentic service' certificate, but the remaining data transfer could contain a fake service' data. All data for a given service must be authenticated to avoid any missing links that could be misused by an attacker.

- When the computer communicates with the OffPAD, you trust the computer to behave correctly. If the computer is compromised, it could

try to trick the OffPAD into for instance handing out credentials for services. The OffPAD must protect against compromised computers in that the user must validate and accept all communication and services should be easily identifiable by the *Pet Name system*. The user is in danger of phishing attacks, where the OffPAD is asked for confirmation about a service that only *looks* like the service the user actually wants, but the URL might for instance be similar enough for some users to not see the difference. The OffPAD must have tools for protecting against phishing scenarios and compromised computers, where the *Pet Name system* is one such tool.

There are attacks on the PKIX that will not be protected by the OffPAD.

9.2.2 Managing the CA certificates

For the OffPAD to be able to do PKIX validation, it will need a storage of all the CA certificates that it should trust. The list of certificates must be chosen wisely, and the number of certificates should be kept as low as possible while still being able to authenticate all needed services. The risk of adding more certificates is that one of the CAs might get compromised and could give an attacker a certificate for any chosen site. The risk of adding less certificates is that some authentic services might not be authenticated by the OffPAD, and the user would therefore not be able to use this service, which would mean that the service would lose money and the OffPAD would lose reputation.

Intermediate certificates should idealistically not have the need to be pre-stored, as they should be given to the device together with the services' certificates. As long as the intermediate certificate is valid and authenticated by a registered CA in the OffPAD, it could still be cached on the device for faster authentication processes. Unfortunately, some services might be dependent on intermediate certificates for authentication, but the TLS or SSL software for the service might be set up in such a way that it would not transfer intermediate certificates in its TLS handshake, which would block the service from getting authenticated by the OffPAD. If a service is not giving its intermediate certificates, it might be needed by the OffPAD to also store intermediate certificates by default, as long as they are authenticated by a CA certificate. There is no risk involved storing intermediate certificates, as long as they are signed by a CA certificate in the OffPAD, except for the extra work required by the software developer of the OffPAD.

The CA certificates must be given to the OffPAD in a way that would not be easily manipulated by attackers. The OffPAD could, to start with, come pre-installed with all the needed CA certificates. The end user must trust the code that is installed on the OffPAD anyway, which means that the CA list will not require further trust. Even if the user is installing the firmware

on the device, it must also trust the OffPAD's hardware, which could contain executable code that is not easily reachable and adjustable. Since the user must trust the device, the CA certificates could also be trusted as they come from the same entity.

The user should be able to disable any of the CA certificates in the OffPAD's settings. It is the user's choice to not trust a CA, e.g. after the CA has reached the news of getting attacked. The list of CA certificates should be presented in a way that is easily understandable for the user. The user should, however, not be able to add new CA certificates manually. Most users does not know how to authenticate a CA certificate, and an option for adding new CA certificates would create the risk of phishing attacks. The user could get asked of manually adding a CA certificate «due to technical problems». A solution for giving the user the option to add new CAs is that some of the CA certificates could have been installed, but disabled by default, which the user could then activate if wanted. By activating certificates we are still in risk of phishing attacks, but now the certificate is at least authenticated properly before activation.

The computer or device communicating with the OffPAD will not know what CAs the OffPAD trusts. A given service might be authenticated by the computer or device, while the OffPAD does not accept it, which must be a valid situation for the computer or device to handle. If the computer or device should not authenticate a service, it should stop the process before it reaches the OffPAD, as there is no point in forwarding it to the OffPAD. The computer or device has more data it could check when authenticating, for instance the IP address, the service' TLS settings or if *certificate pinning* is set for the service in the browser.

The list of CA certificates must be updated regularly, as CAs get compromised, new CAs get created, some CAs might change their public key and come CA certificates needs to be recreated due to their end date. The update of the CA certificates must happen through a secure extra-protocol channel to avoid attacks where the OffPAD could get injected by root certificates. Some solutions for updating the OffPAD's list of CA certificates:

- When connecting to devices, like the end user's computer, the OffPAD could be checked for updates of the firmware and other software. In the same go, also the CA certificate list could be updated. It is important that the firmware that gets downloaded into the device is signed with the manufacturer's public key, which the OffPAD must have pre installed, before the update could proceed. The OffPAD must make sure that the manufacturer's specific public key is used, and not any certificate that is signed by a CA.

  The update of the CA certificate list should happen automatically, as the user might not want to accept it if the OffPAD would be asking for

confirmation before updating it. The end user might not know what a CA or a certificate is, and we are in danger of never updating the CA list. The user should still, however, be informed about the update, and given the possibility to disable new certificates.

One possible attack on the OffPAD is to block the update of CA certificate list, so that a fake certificate could be used. If the computer is compromised and wants to make use of a fake certificate, it would not want to give the OffPAD an updated certificate list which might remove the compromised CA. The OffPAD is dependent on the device it communicates with for getting online information, so it would not have any other channels to get updated.

One solution for forcing updates of CA certificates would be to disable all certificates after a certain period of not getting updated. Requiring updates periodically must be weighed against the risk of a shop not being online or the server that hands out the certificate updates being temporarily down. A challenge for the certificate update is that when a CA is compromised, the certificates would be misused as fast as possible. If an update happens regularly, e.g. once a week, the damage might have already been done, and if the update is required too often, the OffPAD would be totally dependent on the certificate update servers, which would also get much more load. The update time is a known challenge for the PKIX regime [16], and a solution would be to use a revocation lists protocol to have some protection against attacks on certificates.

- The manufacturer, or one trusted party, could be validating all accepted certificates by signing them with their own public key, which must be pre-installed on the device. When authenticating a service, the OffPAD must, in addition to validating the chain of intermediate certificates that should be sent together with the server certificate, also validate one more entity, the last signature of the CA certificate that is signed by the single trusted third party.

  In cases where a certificate's key has been compromised, a plausible attack would be to block access of updated signatures where the compromised certificate is not present. To prevent such an attack the public keys might have a limited lifetime from a few seconds to up to a day. The OffPAD would be required to regularly check online for new CA signatures for working with short lived signatures.

In both situations, the OffPAD must trust at least one third party, either the manufacturer or the software supplier. Trusting a third party is a risk, but the end user is already taking that risk by trusting the hardware of the OffPAD, and also the software of the device if it has not been installed by the user itself.

Using a revocation protocol should be used in addition, to avoid that the OffPAD uses compromised certificates. This, however, requires an infrastructure for the revocation protocol. The CAs have had a tendency to not publish its compromised certificates, but rather been secretly about it. [16] The CAs would need to change their behaviour for revocation updates to be usable.

9.2.3 The PKIX authenticating process for the OffPAD

The process of authenticating a service through PKIX should follow the specification in RFC5280 [10], but with the extra layer of the OffPAD communicating with a device, such as a computer, for getting online information.

1. The computer or device must authenticate the service through a regular TLS handshake, as defined in RFC5246 [11], and regular PKIX authentication as defined in RFC5280 [10]. The browser might also add other constraints to the authentication, like *certificate pinning* and the *HTTP Strict Transport Security policy*, as defined in RFC6797 [18]. Extra authentication mechanisms are only available to the browser client and not the OffPAD unless we explicitly transfer such information to it.

   If the authentication fails, the device must stop here and notify the user. There is no point in involving the OffPAD with a service that we already know is not authentic.

2. The computer needs information from the OffPAD, e.g. user credentials for a service.

3. The OffPAD is activated by the user.

4. The computer hands over information about the service for authentication:

   - The service' certificate.
   - All intermediate certificates, either given by the service or cached by the browser.
   - The full URL of the service.
   - Information about what is requested, for instance the credentials for the service. The request might also include service data, like the challenge for a challenge-response protocol.

5. The OffPAD processes the information and authenticates the service following the specification of RFC5280 [10], the authentication type $\mathbf{S} \rightarrow \mathbf{C}$:

(a) All the attributes for the service' certificate, especially the *valid period*, *algorithm*, *version* and *serial number*, must be valid.

(b) The *fingerprint* of the certificate must not be in a revocation list.

(c) The signature of the certificate must match the issuer's signature, which might be an intermediate certificate or an enabled and valid CA certificate. If the signature does not match any certificate the authentication process fails.

(d) If the issuer is an intermediate certificate the authentication process restart for that certificate, until the issuer of a certificate is an enabled and valid CA certificate.

(e) The *Subject name* of the service' certificate must either match the host name in the given URL, as given from the computer, or it must match one of the names in the extension for alternative subject names, if set for the certificate.

If the service could not be authenticated, an error is returned to the computer or the device. The user should not be able to force through the request.

6. The OffPAD must ask the end user if the service could be recognised for the *cognitive server authentication* $\mathbf{S} \rightarrow \mathbf{U}$. Here the *Pet Name system* could be used for making it easier for the user to recognise and authenticate the service.

7. The user is asked if the given request should be accepted.

8. The OffPAD should respond with the information that was requested to the computer or device, for instance user credentials for the service. If the given request supports it, the returned information should be signed or encrypted for the given service.

In the process both the OffPAD and the user authenticates the given service, and the computer gets the response from the OffPAD to forward on to the service. Note that the given request should contain authentication data from the service, or the response should only be readable by the service, e.g. by encrypting it with the service' certificate, to avoid both passive and active attacks from a compromised computer.

9.3 Authenticating services through TLSA

The OffPAD should follow the specification in DANE'S RFC6698 [19], which describes how HTTPS servers should be authenticated through DNSSEC. The algorithms should work in the same way as the description, with the exceptions as described further on.

Our OffPAD could make use of the TLSA Resource Records in the same way as the RFC describes how web browsers should make use of it. It could then authenticate the servers it should get data from and send transactions to. It could also make use of the `CERT` RR from RFC4398 [28] or the `TXT` RR from RFC1035 [38], when it needs other certificates to validate, for instance if a company does not use the same certificate on its transactions than on their web pages, but this must be standardised before usage and is not covered by this thesis. All the Resource Records must still be located in the DNS data for the server that it communicates with, for the device to be able to authenticate it.

For the OffPAD to be able to use DNSSEC for server authentication, it needs to have the DNS root's public key installed internally. This key is in DNS used to sign the top of the chain, followed by other entities' public keys. Other keys could be imported to the OffPAD and authenticated by the root key, but the root key itself must be pre installed and can only be swapped out in a secure extra-protocol channel.

Since the OffPAD does not support regular online protocols by itself for receiving authentication data, it can not ask DNS by information directly. The data must be given to the OffPAD through the online device it communicates with and handles the transactions. A computer or a shop's device must be considered a threat, as it could be tampered with, or it could be an attacker's device. We must define how this data should be given, what trust anchors to use and how the OffPAD should respond to different input, to avoid security holes.

9.3.1 Attacks on TLSA

Note that the TLSA standard, RFC6698 [19], suggests that stub resolvers[1] can support DNS-based authentication, as long as they trust their recursive name server, and that they communicate with it through a trusted channel. This behavior should not be supported by our OffPAD, due to the idea that the user should be able to fully trust the OffPAD. By making a recursive resolver do the DNSSEC authentication, we must trust this third party for giving us the correct information. We want to trust as few third parties as possible. The OffPAD can not trust the online device or computer's DNSSEC authentication process either, as the attacker could have full control of the computer or online device. To fully authenticate a service the OffPAD must authenticate the DNSSEC data and the TLSA data in the full recursive manner.

The TLSA standard defines that if either the DNSSEC validation status, as defined in RFC4033 [1], is *bogus* or *indeterminate* the TLS connection should be aborted, if TLSA authentication should be used. The statuses

[1]Resolvers that relies on their name servers to find the authoritative DNS data for them, instead of querying all the nodes downwards themselves.

could occur if for instance DNSSEC authentication data were not given or if the DNSSEC authentication process failed, or if the TLSA resource record contains bogus, unreadable data. Only in the case were zero usable certificate associations were given from DNS or the service' DNS zone does not support DNSSEC should the client fall back to the PKIX authentication. This is to avoid attacks were DNSSEC data or TLSA data is blocked by an attacker to force the client to fall back to PKIX authentication where a vulnerability could be exploited more easily.

Due to the variety in usage of TLS, the standard also specifies that client could behave differently depending on *local policy*. We have two options for how to use TLSA:

- Force using TLSA, and only fall back to PKIX if DNS tells us that the service is not set up with TLSA authentication.
- Try to use TLSA, if set up for the service, and inform the user if TLSA is in use or not, e.g. by displaying an icon on the screen when the cognitive service authentication occurs.

When considering the security usability principles, informing about the status of TLSA or DNSSEC will not make sense for most users, and the users will not know if they should take action or not. In stressful moments they would most likely simply ignore this information and proceed with the transaction. Forcing the TLSA authentication, if set up by the service, would make less choice and a simpler process for the end user. The downside of forcing TLSA, is that a service could get blocked for various reasons:

- The DNSSEC communication could get blocked, either forced by an attacker or wrongly set up hardware, for instance local routers that is set up by only allowing UDP communication on port 53.
- The service's administrators might have set up the TLSA record wrongly, making it unreadable by clients.
- Bugs in the online device's resolver might give the OffPAD bogus DNS information.

Errors that could lead to blocking services should be compared against the security usability and the security for the user for if TLSA should be forced or not. By not forcing TLSA, we makes it easier for an attacker to block DNSSEC and make use of the vulnerabilities in PKIX.

9.3.2 Validating DNSSEC chains in the OffPAD

The OffPAD should support authenticating different kind of data from DNS to support the different use cases. It should be able to authenticate the TLSA resource records, but also the more generic `TXT` and `CERT` records. For

the DNS data to be authenticated, the OffPAD would first need to get the required data from DNS, together with all the DNSSEC resource records necessary for validating the chain of trust. By using the pre installed DNS root key, it would be able to authenticate the DNS data.

Data from DNS that could not be authenticated by the DNS root key should be considered unusable by the OffPAD. Note that many zones, even many top level domains, does not have a `DNSKEY`, which means that their hosts' DNS data can never be authenticated and used by the OffPAD. The routines in the OffPAD that makes use of DNS data must take the lack of signed zones in consideration, and might support fall backs for such services. It is important that the routine gets an *authenticated* response of missing `DNSKEY`, by `NSEC` and `NSEC3` RR. An attacker might try to block the DNS data from getting into the OffPAD, which should not trigger fall back mechanisms, but instead just block the routine instead. When the following text mentions the `NSEC` RR, it also includes the `NSEC3` RR.

Note that Appendix A in the TLSA standard [19] notes that TLS clients validate certificates differently, and depending on what is set to be validated in TLSA - either the full certificate or the SubjectPublicKeyInfo - one could either be blocked by valid certificates or let some types of invalid certificates get through. One therefore need to know how the TLS client is working internally to be sure of what kind of validation to use. Most web browsers use TLS in about the same way, as they are equally interested in getting access to the same services, but other TLS clients could behave differently.

Getting the DNS data for the OffPAD

Since the OffPAD is not online, the data from DNS has to be given to it in a format it could understand. To be able to validate that the data is authentic, the DNS resource records has to be given to it unmodified, or else the signatures would be invalid and could not be verified. The OffPAD uses its pre installed DNS root key to validate the data, and needs the DNSSEC resource records from the root zone to the given host's zone.

What is needed by the OffPAD from the online device:

- The OffPAD needs to know the host name and port number of the service. To later be able to support other usages, the OffPAD would also like to know if TCP or UDP is used in the communication with the service. This is used to know and verify the location of the target resource records.
- If a zone contains a `CNAME` RR, the chain must continue at the reference.
- All the target RR that matches the service' location. The RR type could be `TLSA`, but other types could occur for authentication of other usage, like `CERT` or `TXT` RR. Note that an `NSEC` or `NSEC3` RR must be given instead if no target record could be found for the specific service.

- The `RRSIG` RR for the target record.
- The zone's `DNSKEY` RRs. The zone might contain more than one key, in which one key sign the other hierarchically. All the keys must be given, with their `RRSIG` RR.
- The parent zone's `DS` RR, to verify the given zone's `DNSKEY` RR, together with the `RRSIG` RR.
- The rest of the DNSSEC RR types in the chain of parent zones until you reach the DNS root key:
  - All the given zone's *DNSKEY* RRs. The client does not differentiate between zone signing keys and key signing keys, as they are all needed to verify the signature chain.
  - The parent zone's *DS* RR, together with its *RRSIG* RR.

If an `NSEC` or `NSEC3` RR is returned instead of any of the DNSSEC specific records, the target record could not be authenticated. If some of the resource records is missing, and no `NSEC` record exists instead of it, the whole routine should be aborted, to prevent an attacker exploit fall back mechanisms. The user should not be able ignore such errors and carry on with the requested transaction.

All the given RRs and DNS signatures must be in their valid lifetime. If any of the records are out of date, it should be considered invalid and be ignored. Ignoring any of the records would most often lead to the whole routine getting aborted, as an `NSEC` or `NSEC3` RR would be missing.

The data communicated for certificate validation does not need to be encrypted between the OffPAD and the online part, if one wants to save time and processing power. The data is already integrity protected by the signatures in DNS, and the given data is already publicly available. Privacy might be a valid reason for encrypting the data, as anyone near the communication channel could easily find out what domains the OffPAD is interested in. One might also, if the OffPAD would not receive already cached DNS data, see what host names the OffPAD have already cached by the lack of communication.

One related issue is collision attacks upon the certificate signature hash values, if that is stored in the RR - i.e. if the Matching Type Field is set to SHA-256 or SHA-512 an attacker could try to generate a fake certificate that has the same hash value as the original certificate. This is not an issue today, as these hash algorithms are considered strong enough to not be practically brute forced.

The format of the DNS data

The DNS data is by resolvers fetched through network packets, and each packet is for a separate RR. The OffPAD should receive each RR that is needed in full in the communication. All the needed records could be given in one communication packet if that makes the communication with the OffPAD faster, but we must be able to separate the records when reading the DNS data.

Google's solution, as mentioned in Section 5.7.3 on page 45, is an interesting approach, which has both an easy solution, a defined and compact data format and existing code. The data given in the certificate extension gives us all the necessary data the OffPAD needs to fully validate the DNSSEC chain of trust and authenticate the service by its `TLSA` RR. The certificate itself is not necessary for the OffPAD, as it gives us no extra information except for a binding between the service certificate and the DNS data, but the binding does not contain any authentication since the certificate is self-signed and could be generated by anyone.

In Google's solution for service authentication through DNSSEC, the server generates the data that should get transferred. The OffPAD would get the same benefits as regular web browsers, as mentioned in Section 5.7.3 on page 45, but to avoid some of the downsides, we do not need to require that the service' certificate itself is changed. As long as the service' administrators update the `TLSA` RR for the service with the correct certificate, and the DNSSEC data is given to the OffPAD, we are able to authenticate the service. By separating the DNSSEC data from the service certificate would let services be able to use any certificate they want, either signed by a CA or self-signed, as long as the OffPAD also gets hold of the DNSSEC chain for the authentication process to succeed.

To avoid that each online device that communicates with the OffPAD have to implement the functionality of a recursive resolver, we could either, as in Google's solution, let the server generate the dump of DNSSEC data, or it could be separated out to third party servers. The OffPAD must validate the DNSSEC chain of trust, which means that we do not put our trust in the servers that generates the DNS dump. Advantages of outsourcing the generation of DNSSEC dumps, if a standard protocol is used:

- The DNS generation service could be used for all services which has a `TLSA` RR registered, as the server could generate a DNSSEC dump for every publicly available DNS node.
- The DNS generation service could be used by many different clients, as long as it uses a standardized protocol.
- The generation of the DNS dump could easily be spread out on many different servers, and even different vendors, which would improve stability and faster response time for the clients.

- The servers could cache the DNS data and the DNS dumps, which would give less load on the DNS resolvers and faster response time.

By defining how one server should support generating the DNS dump, it could later be expanded with more servers. The online devices would only need to know the updated URL of where to retrieve the DNS dumps for services.

In contrast to Google's solution, we would also need to include `NSEC` and/or `NSEC3` RR, to be able to support PKIX authentication for services not using TLSA, without the vulnerability of downgrade attacks, as mentioned in Section 9.6 on page 88.

The dump from Google's solution is in the *c struct* format, which is small and efficient, and there exists libraries for different programming languages for reading and writing in this format. The OffPAD should be able to work with the *struct* format, but other formats could be used if the existing format is a problem.

In case of missing DNS settings

What should happen if the OffPAD does not get any DNS data when validating a host, or if the given DNS data does not validate? It could either fall back to the old X.509 authentication, which would greatly reduce the security, or it could block the host until it gets proper information from DNS. The former solution will make the OffPAD vulnerable for every attack upon X.509 which has been mentioned in the introduction of this document, thus reducing the security to a much lower level. This solution would even make this whole thesis obsolete, as it is easy to block parts of the communication with an OffPAD and will therefore also make it too easy to make it work in fall back mode.

If the OffPAD can not get proper DNS data for a host it is best to simply block this host. The drawback of this solution is that users would not be able to accept transactions for authentic hosts if there are problems with the communication with the DNS name servers, or the online device does not give it the DNS data properly. It is however still better than being open to X.509 attacks.

Instead, if a server does not have any certificate settings, the online device must give the OffPAD an *NSEC* or `NSEC3` RR. This gives the OffPAD an authentic answer that the server does not specify any certificate settings, and it should then use the old PKIX authentication chain.

However, if a server is missing data in DNS, but an `NSEC` is returned instead, the OffPAD could fall back to using PKIX for authenticating the certificate. If it does not authenticate the certificate through PKIX, even self signed certificates could be used, and the OffPAD would be fine with that. Even though PKIX would not stop an attacker, at least it makes it a bit harder to attack a device.

The process of authenticating the chain of trust in DNSSEC

The process of authenticating the chain of trust in DNSSEC in the OffPAD must implement the exact same routine as a security-aware DNS resolver, as defined in RFC4035 [2, section 5], but with the difference in that the information should not be requested online, as all information is expected to already exist locally. See Section 5.6.2 on page 40 for details in the authentication algorithm.

The OffPAD needs only the DNS root's public key for validating the DNS chain of trust, but to speed up the processing, already given DNS nodes' keys could be cached until either the RR's *TTL* or `RRSIG` expires. Every `DNSKEY` RR has to be validated before being cached.

Managing the DNS root key

In case the DNS root's keys being compromised, it should be possible to update the trust anchors in the devices. This has to be done in a secure manner, through a secure extra-protocol channel, to avoid that the OffPAD could get attacked by modifying the stored DNS root key. If the security for updating a device's trust anchors is not strong enough, it should rather not be possible to update them.

Since the end user of the OffPAD already have to trust the hardware, firmware and software on the device, and then also the manufacturer and the software developers, this is a usable entry point for updating the trust anchors. The OffPAD could have its trust anchors updated through regular firmware updates, as long as the firmware is fully authenticated before getting installed.

Note that the authentication of the manufacturer should *not* happen through PKIX authentication, as that would leave the OffPAD open to PKIX vulnerabilities and make the TLSA authentication worthless. Since PKIX authentication is considered weaker than TLSA authentication, an update of the trust anchors must either happen with TLSA authentication or an other form of authentication. An example on another form of authentication that could be used for updates is the use of regular public-key cryptography, without any form of PKI and third parties that must be trusted. By using regular public keys, the device would *only* accept firmware updates if they're signed with the manufacturer's private key, that matches the pre stored public key in the device.

The user could have the possibility to manually add the DNS root key or other DNS trust anchors, by typing it directly into the OffPAD. It is then up to the user to be able to verify the trust anchor. By supporting a manually update of the DNS trust anchors, we give the user more control of the device, but we are also opening up for phishing attacks in which the user is tricked into adding a trust anchor for a fake DNS name server. Manually user input

could also be combined with the firmware updates, in which the user could type in the hash of the trust anchor before it gets accepted and installed on the OffPAD.

9.3.3 The TLSA authentication process for the OffPAD

The process of authenticating a service through TLSA should follow the specification in RFC6698 [19], but by using the online device or computer as an untrusted resolver for getting the information. The process must also support regular PKIX authentication, as described in Section 9.2.3 on page 73, both for certain TLSA settings and for fall back.

A requirement for making use of TLSA, is that the computer that communicates with the OffPAD supports TLSA as well. If a service is set up to only authenticate through TLSA with its given certificate association and skip the PKIX authentication, a computer without support for TLSA would not be able to authenticate the service. A computer or online device should either be able to authenticate a service through TLSA, or it should let the OffPAD authenticate the service. The online device could not authenticate the service only through PKIX, to avoid services getting blocked.

The process for authenticating with TLSA:

1. The computer or device must authenticate the service through a regular TLS handshake, as defined in RFC5246 [11], in combination with TLSA, as specified in RFC6698 [19]. If the computer or online device does not support TLSA authentication, it needs to trust the OffPAD for authenticating the service in full when the request is given.

2. The computer needs information from the OffPAD, e.g. user credentials for a service.

3. The OffPAD is activated by the user.

4. The computer hands over information about the service for authentication:

    - The service' certificate.
    - All intermediate certificates, given by the service through the TLS handshake.
    - All TLSA resource records or an `NSEC` or `NSEC3` RR if no TLSA is defined.
    - The full DNSSEC records of the types `RRSIG`, `DNSKEY`, `DS`, from the root node and down to the service zone. If the path is changed by a `CNAME` RR, it must be included as well.
    - The full URL of the service.

- Information about what is requested, for instance the credentials for the service. The request might also include service data, like the challenge for a challenge-response protocol.

5. The OffPAD processes the information and authenticates the service. This is the authentication type **S** → **C**:

   (a) The DNSSEC resource records from the root node to the host's `TLSA` RR must be authenticated, and is described in Section 5.6.2 on page 40. If the DNSSEC authentication could not be validated, the authentication process fails, and the OffPAD returns an error to the online device. If one of the DNS zones does not support DNSSEC, an `NSEC` or `NSEC3` RR should be returned and authenticated, and the OffPAD should fall back to PKIX authentication as described in Section 9.2.3 on page 73.

   (b) The TLSA resource records must be checked for validity. The lifetime of the resource record must be valid, and the *Usage*, *Selector* and *Matching type* fields must contain an acceptable value. Resource records not accepted gets ignored.

   If no `TLSA` RR was registered, the OffPAD falls back to authenticating the service through PKIX, as described in Section 9.2.3 on page 73.

   (c) Each TLSA setting is processed, and matched against the given service certificate.

   - If the TLSA record's usage type is *type 3 - domain-issued certificate* and the given service certificate matches the TLSA's *certificate association data* the authentication process succeeds.
   - If the usage type if on of the types from 0 to 2, the certificate chain must be authenticated, as in PKIX. The TLSA record's *certificate association data* must match one of the certificates in the chain, depending on what the usage type of the TLSA defines. If the constrained certificate chain is valid, the authentication process succeeds.

   If the service could not be authenticated, an error is returned to the computer or the device. The user should not be able to force through the request.

6. The OffPAD must ask the end user if the service could be recognised for the *cognitive server authentication* **S** → **U**. Here the *Pet Name system* could be used for making it easier for the user to recognise and authenticate the service. The OffPAD could for instance map the domain name to a petname and present the given petname to the user.

7. The user is asked if the given request should be accepted.

8. The OffPAD should respond with the information that was requested to the computer or device, for instance user credentials for the service. If the given request supports it, the returned information should be signed or encrypted for the given service.

If NFC would be used for the communicating part between the online device and the OffPAD, it would have a top speed of $6780Kb/sec$ in active mode. [44]. When considering that a normal DNSSEC chain would be of at least $2.5KB$ [34], it would idealistically require

$$\frac{2.5KB}{\frac{6780Kb}{second}} = \frac{20Kb}{\frac{6780Kb}{second}} = \mathbf{2.95msec}$$

to transfer the extra data from the online device to the OffPAD. In most cases it would take longer. The time it takes for the online device to retrieve the DNSSEC data, and the time for both the online device and the OffPAD to process and authenticate the data comes in addition.

9.4 User involvement

The OffPAD must be able to full fill the *Security Principals* for user involvement as defined in Section 5.2 on page 16 when the DNSSEC authentication is in progress. There is here a distinct difference between the syntactic server authentication process using DNSSEC and X.509, and the cognitive server authentication that could come afterwards. The former process is purely technical and does not focus on usability, while the latter, the cognitive server authentication, is designed to be used and understood by humans. It is the syntactic part of the server authentication that is discussed in this section, we already know that the user *must* be involved in the cognitive authentication process, $\mathbf{S} \rightarrow \mathbf{U}$.

The cognitive server authentication could for instance be using the *Pet Name* system for making it easier for the user to recognise the services.

The server authentication process has many steps, and all of them could have a different outcome, depending on the policy of the client. Policies, or decisions, could also be decided through user involvement, as is seen in web browser when a self signed server certificate is used for a web site - the user is asked to accept or deny the certificate. The main situations in the server authentication process where user involvement *could* be given:

1. The OffPAD is asked to get involved with a service. The user could get asked if the OffPAD should be authenticating the service through DNSSEC or X.509, or both of them. The transactions might involve many servers, so the user could even decide this for each of the servers.

The user could even decide if the server authentication should not happen at all. It would not be a recommended decision, but it is still a plausible decision.

2. The OffPAD has asked the computer about DNS data, but has not received any reply. This could either be a technical problem, for example if DNSSEC is blocked in the local firewall, or it could mean that an attacker is blocking the data to prevent the OffPAD from discovering that the server is fake.

   The user could decide if DNSSEC authentication should be skipped or not. A timeout period could also be set for this scenario.

3. The DNSSEC data could be invalid, due to missing, expired or corrupt data. This could either be a setup error by the administrators of the service or registrars or a software bug, but it could also be an attack attempt.

   The user could decide to skip DNSSEC authentication and only go for X.509.

4. The DNSSEC data could give the OffPAD an `NSEC` or `NSEC3` result instead of a `DNSKEY` RR, meaning that the server's zone, or a parent zone, is not signed by DNSSEC and could therefore not be authenticated through DNSSEC.

   The user could decide to skip DNSSEC authentication and go for X.509 instead.

5. The DNSSEC data could give the OffPAD no *TLSA* records for the service, and instead return an `NSEC` or `NSEC3` RR. This could either mean that the service is not using TLSA, or the server administrators could have mis configured the server. An attacker would either be discovered in the previous situations mentioned over, as the DNSSEC keys would not match, or it would have gotten access to DNS, having full control of all the data anyway.

   The user could decide to skip DNSSEC authentication and go for X.509 instead.

6. The *TLSA* records for the service could all be invalid, leading to no records that could be used for the authentication. This could either be a software bug or a human error by the DNS administrators. In either case, the OffPAD would not be able to authenticate the service through DNSSEC.

   The user could decide to skip DNSSEC authentication and go for X.509 instead.

7. There could exist *TLSA* records for the service, but none of them would match the server certificate. This is where the TLSA specification [19] tells you that the service is not authenticated and further communication with the service should stop.

   It would not be very wise to still accept the server when this happens, but the user *could* still be able to either stop the process, skip the authentication at all, or try X.509.

8. The matching `TLSA` RR could give various settings on how the certificate should be validated. Some of the settings might not be acceptable for the OffPAD or the end user, for example if the certificate in addition should be authenticated by X.509 or not, or it could give its own CA certificate to use in the authentication process. The TLSA specification mentions that client could have local policies that could say otherwise. [19], meaning that clients could override the settings from TLSA, if the local policy says so. Normally, you would not like to change the way the server itself asks about authenticating itself, as long as you trust DNSSEC, but it is still a decision that could be taken by user involvement.

Starting with the *Security Conclusions* in item 2 on page 16, item *2a* requires that the user understands enough about what is asked for, and its consequences, to be able to make an informed decision. If a normal user should get involved in the decisions of the syntactic part of the service authentication, it would need to know about what kind of phishing attacks, network attacks and other types of attacks that could occur in the different use cases. The service authentication process depends on up to two PKI systems, the DNSSEC and the PKIX, with the TLSA standard as a binder between them. The PKI systems and the different protocols contains a lot of details.

The conclusion is that the OffPAD must not involve the user at all in the syntactic server authentication process. The user is not qualified to make secure decisions, and we are not able to give the user enough information that is mentally applicable. The OffPAD must either accept or deny the outcome of the authentication process, and not give the user any choice in the process of authenticating a server through DNSSEC, as that could lead to successful attacks on the OffPAD. Instead, the user should be focused on the next step in the authentication process, the cognitive service authentication process, where the user must accept or deny a service for example through the Pet Name system.

However, some settings could be set by the user in the configuration of the device, when not in the middle of an authentication process. An example of a configuration setting is if the certificates should be validated through X.509 in addition to DNSSEC, or only using DNSSEC, thus overriding services'

TLSA settings. The reason for accepting this, is that when outside of the authentication process, the user has more time to think through the decision. Before changing security settings, the user could gather more information, either from the device' manual, or consult people that knows about the risks involved.

An essential point that must be considered when deciding the user involvement in the authentication process, is that the user is often stressed for decisions there and then, e.g. when in a shop, and the mental capacity is therefore low. The user could easily get distracted or interrupted by employees, children or other people, and the user might be in the front of a queue of people waiting for the user to finish, which could all raise the stress level. In the worst case, the user could accept any decision just to be able to finish off the process as fast as possible. Having as few and easy decisions as possible is crucial for the security.

9.5 Changes for the service administrators

For a service to be able to use TLSA for authentication, the web site must be hosted by a registrar that supports DNSSEC. Also, the administrators of the service would need to be able to put the server's certificate into DNS. The service itself is not affected by changing to TLSA, other than being set up with other certificates, e.g. self signed ones.

9.5.1 Roll out new certificates and keys

When using DNS for storing server certificates, one has to change the routine when updating the server's certificate. The certificate has to be put into DNS some time before the server is updated. After the server is updated, the old certificate could be removed from DNS. This is different from the routine in PKIX.

How to roll out new certificates is described in Appendix A.4 in RFC6698 [19]. It is important that the new certificate gets added to a DNS RR some time before it gets used by the server, since the DNS data is cached. In TLSA, you are allowed to have many records for the same service, and the standard defines that the client must check each RR until it finds a certificate match.

9.5.2 Revocation

If a TLSA for a service is set up to ignore the X.509 chain, clients would also by default ignore revocation lists for X.509 certificates. This means that the domain administrator gets more responsibility for checking that their certificates are trustworthy and update their server and/or DNS. This depends on what TLSA settings the service is using.

The risk of a compromised certificate is that the OffPAD could successfully authenticate the certificate for an attacker. If the certificate is self-signed and TLSA is used, the OffPAD would still only accept the certificate for the given service. The attacker would also have to get control of the DNS settings of the host for active attacks.

Depending on the TLS settings an attacker would be able to take over the encrypted connection, since the certificate's public key is used to encrypt data sent to the service. If no key exchange algorithm like Diffie-Hellman is used, but the shared secret key for further symmetric encryption is encrypted with the server's public key, the attacker would be able to both listen to and modify the encrypted transaction.

9.5.3 Challenge-response authentication

To improve the integrity of the service' credentials, the authentication process would happen through a challenge-response protocol, like the *Extended HTTP Digest Access Authentication* protocol, instead of sending credentials like username and password. The browser is able to see the username and password, and could be saving it for later, while for a challenge-response protocol, it is only able to see the challenge and the response, and not the secret required to generated the response.

The problem for the OffPAD is that there is no binding between the challenge and the service. If the OffPAD gets a challenge for a service, it cannot authenticate where the challenge request came from, as the challenge itself is not signed. A compromised computer could send a request, pretending to be one service, but would instead send its own, modified challenge, to try to find the secret that could generate challenge-responses.

9.6 Threats and other security considerations when using TLSA in the OffPAD

The whole proposal depends on the security of DNSSEC and its chain of trust. Even though the chain is cryptographically secured, the entities that administrates their local DNS data, e.g. registrants, are still vulnerable to attacks that in the end could affect the domains that the OffPAD is communicating with. We move the threat from vulnerabilities in one of the many CAs, to a vulnerability in one of the DNS zones in our direct chain upwards to the root node. Some DNS companies regards security issues as critical, others might not be that protected. At least the total number of nodes that could be attacked has in most cases been lowered from the PKIX.

9.6.1 Threats from communicating devices

The OffPAD is doing the authentication by communicating with the client terminal, which can be a personal computer, a smartphone or a payment device in a shop. Such a system could be tampered with, and could for instance be trying to give the OffPAD the attributes of a totally different service than what was actually requested.

In such scenarios the OffPAD could be successfully authenticating the other service, as it is a proper service, even though the client platform is compromised by an attacker. It would not be able to get the credentials for the proper service, as the host names differ, and the OffPAD would then block the response.

Another attack scenario is phishing attacks, where the user gets connected to a valid service, but not the service that was thought of. Most of such attacks are not something the PKI could prevent against, but the OffPAD should have tools for this, such as the Pet Name system.

9.6.2 Downgrading

A way to attack an authentication protocol is to block certain data to be able to downgrade to a older, more insecure protocol. When the initial transactions are sent, the attacker could either block the communication, or change it to contain bogus information, thus leading the device to think that the service does not support the newest version.

The type of attack could be attempted when a web site using TLSA should be authenticated. The data transfer could easily get blocked when the DNS data should be given, in which the OffPAD has the X.509 certificate for the service, but not the TLSA and the chain of trust from DNSSEC. It could be tempting to fall back to authenticating the service through PKIX, but this would make it easy to attack the OffPAD by downgrading it. The extra protection put in TLSA will be of no use, as it could easily be turned off.

The data transfer for this attack could be blocked in different places:

- A compromised computer could choose to not send the DNS data. This includes compromised devices in a shop.
- The communication protocol, like the NFC, could be read and blocked when data is transferred. Even if all the data would go encrypted over NFC, guesses could be made to when the data should get blocked, as the amount of information that needs to be transferred does not vary much from service to service.
- The network could have been tampered with so that TLSA resource records could be blocked.

- The DNS resolver could have been compromised to not hand out TLSA data.
- The DNS server could have been compromised and the TLSA resource records could have been removed. This is a more serious threat which the OffPAD can't protect against, since a compromised DNS server could hand out a different IP address for the service, together with a TLSA for the fake service.

For the OffPAD to avoid such an attack, it is important that it does not downgrade to use the older PKIX unless it is certain that TLSA is not used. This means that the information from DNS *must* be given and validated every time a service should get authenticated. It can not simply start using PKIX if no DNS is given, as that would make it easy for an attacker to turn it off.

9.6.3 Differences from X.509

When using X.509, you need to get a signed certificate from a third party, which *should* be authenticating the requester before signing the certificate. When using TLSA, you could create your own, self signed certificate, and tell the clients that this is good enough, not involving any third party.

Even though history shows that it is possible to get false certificates by attacking CAs, it is still not an easy task to do. X.509 is not as secure, but it is raising the difficulty level for an attacker to fake a server. If a server only wants DNSSEC authentication, the attacker has to break into the server's DNS registrar. If a server is using both DNSSEC and X.509 authentication, the attacker has to both be able to change the DNS setup of the server and get hold of a fake certificate for the server.

If an attacker would be able to change the DNS data for a server, and X.509 would be required in addition, the transaction would be stopped as the server certificate would not match, unless the attacker would get hold of a fake certificate for the server host name.

9.6.4 Attacks on third parties

Trusted third parties that is used for signing the services certificates could be attacked. Also, the manufacturer or the device' software entity could be attacked, thus leading to software or firmware updates with code for making it possible to attack the device or get private information from it.

9.6.5 Initial key delivery

The system does not solve the initial key delivery. Even though we do not need all the CAs' public keys, we still need at least one key before we could

start authenticate servers: the public root key. This key must be given to the OffPAD somehow, and is not described in this proposal nor in the DANE documents, except that it should be delivered in an off-band channel.

If a company should produce such an OffPAD, it could ship the devices with a copy of the root key pre installed. It would however be crucial that this key is the correct one, so you would need to make sure that the company got the key in an off-band channel, and that the customers would be possible to see the whole key that is stored on the OffPAD, to verify it themselves.

An off-band delivery of the root key is to travel to the location of the root key or to meet an administrator of the key that you would trust, and get a copy of the key. This does not scale up, as there are not that many root node administrators. You could however bring your root key back, and give it to others that have trust in you. Put in a system, this could be done by central organizations, e.g. national DNS registries like Norid [2] or more local organizations like IT-departments of universities. You could then get an off-band copy of the root key yourself through entities you trust.

The customers buying an OffPAD from a company would need to trust the company for putting the correct code on the OffPAD. Even though the code could be open source, a regular customer would not be able to open up the device and verify that it contains the correct code, especially not if algorithms, like the crypto-algorithms, are implemented in hardware for efficiency. Even though the key could be verified, one might not be able to verify the code and the OffPAD's hardware functions.

9.6.6 Missing link between client software and end user

Even if the server authentication should be correctly validated by the OffPAD, we are still missing the last link, the end user. The solution by this thesis is only authenticating the servers in an improved, technical way, the end user is still not involved. It is, as is mentioned in the theory part of the thesis, crucial that the end user is able to authenticate the service. This is not discussed in this thesis, but is covered in other documents in the Lucidman project, for instance the *Petname system* [15]

[2] norid.no

Part III

Conclusion

Chapter 10

Summary

Authenticating services is a challenge for the security community, especially when it comes to the initial authentication process, or the registration process. The user lacks the tools for making sure that a service is authentic through today's PKIX, which is why phishing attacks are popular. The OffPAD is a device that should give the user more tools for securing its online identities and to authenticate services and protect both from network attacks and phishing attacks. When the user is better protected it will also be beneficial for the services.

The thesis has been focusing on the syntactic service authentication through the use of the more structured chain of trust in DNSSEC, which could be used as a less vulnerable PKI compared to the PKIX authentication. The main use cases for the OffPAD has been discussed, and the thesis has shown that the authentication specification of *TLSA* is beneficial for authentication compared to the PKIX authentication, as it closes some of the vulnerabilities in the authentication process, and is usable for at least the use case that requires authentication with online services.

The *TLSA* authentication process is usable for an offline device such as the OffPAD as long as the DNS root key is pre stored in the OffPAD in a secure and trustful way. The TLSA standard is designed to comply with the different behaviour of TLS clients, and describes how to behave for corner cases. The DNSSEC data that is necessary for the authentication of the servers' TLSA settings is structured and could be standardized to a format that could easily be transferred and authenticated by the OffPAD, using Google Chrome's previous solution. The serialisation of the DNSSEC has the benefit of giving clients the data faster than a recursive resolver, and also be usable in environments where TCP data from DNS is blocked, as described in Section 5.7.1 on page 44. Standardised DNSSEC data could even be outsourced to third party servers, as the data must be authenticated by the DNS root key by the OffPAD before it could be used.

By using the *TLSA* standard for authenticating servers, the OffPAD is

protected against some of the PKIX vulnerabilities. The advantages of *TLSA* are still not as usable as it could be as long as there lacks a cryptographically binding between the service authentication and the request given to the OffPAD - the service request itself might not contain authentication data. Some service requests contains authentication information, for instance a signature, but old fashioned credentials like username and password are not sent from the server with any authentication, since the TLS connection is expected to be the way the data gets authenticated. The OffPAD is not involved in the TLS connection, as it stops at the online device, which means that there is a missing authentication link between the service and the OffPAD for some of the request types.

The cognitive service authentication of the end user is an important part of the service authentication, and usability is important for protecting the user and the OffPAD. Going through the security usability principles has shown that the user should not be involved in the technical service authentication through TLSA, the syntactic authentication process, as this would be too complex to make decisions for when in a transaction. The user must instead be fully involved in the cognitive service authentication.

There are many ways for attacking the OffPAD, and the OffPAD is still vulnerable for some attacks, even though the attack surface is reduced to a minimum by keeping the OffPAD offline. One can never be totally secure, but it is possible to get close.

10.1 Future work

The thesis has mentioned that the OffPAD lacks authentication of the service requests, as described in Section 8.2.1 on page 58. The OffPAD is not able to authenticate e.g. credential requests, which makes it a possible attack scenario, in which a compromised online device could try to get credentials for other services than what it shows to the user. The attack must still be able to pass the cognitive service authentication on the OffPAD, though, as the user must accept the service request before it is given. To be able to solve the challenge of authenticating service requests, the service protocols for the requests might need to be modified. Expanding a challenge-response protocol that also includes authentication, e.g. a signature from the server, would avoid some possible network attack scenarios for the OffPAD but also for other authentication systems, even those implemented in the web browsers.

Another solution that has been thought of for the OffPAD to be able to authenticate service requests, is to let the OffPAD be able to read the TLS data. If the OffPAD is fed with the TLS authentication information, all the symmetric shared TLS keys used and the TLS data that contains the request that needs to be authenticated, it would be able to authenticate that

the request was retrieved from the authentic service. This is however not a suitable approach, as it interferes quite a bit with how TLS works today, as only the sender and receiver should originally be able to read the plaintext. With this approach the OffPAD, or any other, similar device, could become a man-in-the-middle which could exploit the given secret. Even though the OffPAD should be trusted more than the computer or online device, no device should be trusted blindly.

The system for generating a dump of DNSSEC data in a standardized format might be a solution that could be usable for others in addition to the OffPAD. Web browsers require low latency, and getting all the necessary authentication data in one request could be a solution that might help web browsers with getting an easier implementation of TLSA authentication of web servers. The functionality is so generic, and the system is only caching already existing data, so it could be implemented and supported everywhere and by everyone. The DNS serialization could also be set up to be fully redundant. It requires some work and specifications of the service before it could be usable for others, and the challenges that has been mentioned by the DANE work group must be considered to avoid undocumented consequences.

Bibliography

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFC 6014.

[2] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014.

[4] Charles Arthur. Iranian hacker claims he acted alone in stealing digital ssl certificates. *The Guardian*, March 2011. http://www.guardian.co.uk/technology/2011/mar/28/hacker-digital-security-stuxnet-iran.

[5] Charles Arthur. Rogue web certificate could have been used to attack iran dissidents. *The Guardian*, August 2011. https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google.

[6] Internet Assigned Numbers Authority. Dnssec information. https://www.iana.org/dnssec, June 2010.

[7] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.

[8] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP Message Format. RFC 2440 (Proposed Standard), November 1998. Obsoleted by RFC 4880.

[9] ITU (CCITT). Recommendation X.800, Security Architecture for Open Systems Interconnection for CCITT Applications, 1991.

[10] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[11] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[12] Torbjörn Eklöv, Interlan Gefle AB, and Stephan Lagerholm. Operational Challenges When Implementing DNSSEC. *The Internet Protocol Journal*, 13(2):16–26, June 2010.

[13] C. Ellison. SPKI Requirements. RFC 2692 (Experimental), September 1999.

[14] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693 (Experimental), September 1999.

[15] Md. Sadek Ferdous and Audun Jøsang. Entity authentication & trust validation in pki using petname systems. In Atilla Elçi et al., editor, *Theory and Practice of Cryptography Solutions for Secure Information Systems (CRYPSIS)*. IGI Global, May 2013.

[16] P. Gutmann. PKI: it's not dead, just resting. *Computer*, 35(8):41 – 49, August 2002.

[17] Barbara Guttman, Barbara Guttman (au), and Edward A. Roback (au). *An Introduction to Computer Security: The NIST Handbook*. DIANE Publishing, April 1995.

[18] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). Proposed Standard, November 2012.

[19] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), August 2012.

[20] ITU-T. Recommendation X.500 - Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services, 2008.

[21] Audun Jøsang, editor. *Trust Extortion on the Internet*. University of Oslo, June 2011. The 7th International Workshop on Security and Trust Management (STM 2011), Copenhagen.

[22] Audun Jøsang. PKI Trust Models. In *Theory and Practice of Cryptography Solutions for Secure Information Systems (CRYPSIS)*, 2013.

[23] Audun Jøsang, Bander AlFayyadh, Tyrone Grandison, Mohammed Alzomai, and Judith McNamara. Security usability principles for vulnerability analysis and risk assessment. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'07)*, Miami Beach, December 2007.

[24] Audun Jøsang, Mohammed AlZomai, and Suriadi Suriadi. Usability and privacy in identity management architectures. In *Proceedings of the Australasian Information Security Workshop (AISW'07)*, Ballarat, January 2007.

[25] Audun Jøsang and Simon Pope. User centric identity management. In *AusCERT Conference*, The University of Queensland, Australia, May 2005.

[26] Audun Jøsang, Christophe Rosenberger, Laurent Miralabé, Knut Eilif Husa, Jérôme Daveau, and Petter Taugbøl. Local user-centric identity management. Technical report, University of Oslo and ENSICAEN and TazTag and Tellu and CEV and Vallvi, 2013. Draft.

[27] Audun Jøsang, Kent Are Varmedal, Christophe Rosenberger, and Rajendra Kumar. Service Provider Authentication Assurance. In *10th Annual Conference on Privacy, Security and Trust (PST 2012). Paris*. University of Oslo, ENSICAEN, France and Ministry of Communications and Information Technology, India, 10th Annual Conference on Privacy, Security and Trust (PST 2012). Paris, July 2012.

[28] S. Josefsson. Storing Certificates in the Domain Name System (DNS). RFC 4398 (Proposed Standard), March 2006.

[29] Henning Klevjer, Audun Jøsang, and Kent A. Varmedal. Extended http digest access authentication. In *The 3rd IFIP WG 11.6 Working Conference on Policies & Research in Identity Management (IFIP IDMAN 2013)*, London, April 2013.

[30] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[31] A. Langley. Serializing DNS Records with DNSSEC Authentication. http://tools.ietf.org/html/draft-agl-dane-serializechain-01, July 2011. Work in progress, expired January 2, 2012.

[32] Adam Langley. DNSSEC and TLS. http://www.imperialviolet.org/2010/08/16/dnssectls.html, August 2010.

[33] Adam Langley. DNSSEC authenticated HTTPS in Chrome. http://www.imperialviolet.org/2011/06/16/dnssecchrome.html, June 2011.

[34] Adam Langley. Re: [dane] behavior in the face of no answer? http://www.ietf.org/mail-archive/web/dane/current/msg04929.html, May 2012.

[35] B. Laurie and A. Singer. Choose the red pill and the blue pill: A position paper. In ACM, editor, *The 2008 New Security Paradigms Workshop*, pages 127–133. ACM, ACM, 2009.

[36] B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155 (Proposed Standard), March 2008.

[37] Steve Lohr. For Impatient Web Users, an Eye Blink Is Just Too Long to Wait. http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html, February 2012.

[38] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966.

[39] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[40] R. Rosenbaum. Using the Domain Name System To Store Arbitrary String Attributes. RFC 1464 (Experimental), May 1993.

[41] Seth Schoen and Eva Galperin. Iranian man-in-the-middle attack against google demonstrates dangerous weakness of certificate authorities. *Electronic Frontier Foundation*, August 2011. http://www.guardian.co.uk/technology/2011/aug/30/faked-web-certificate-iran-dissidents.

[42] William Stallings. *Network Security Essentials, 4th Edition.* Pearson Education, Inc, 2011.

[43] Mohsen Toorani and Ali Asgar Beheshti. IEEE Xplore - LPKI - A lightweight public key Infrastructure for the mobile environments. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=4737164, 2008.

[44] Kent A. Varmedal, Henning Klevjer, Joakim Hovlandsvåg, Audun Jøsang, and Johann Vincent. The offpad: Requirements and usage. In *The 7th International Conference on Network and System Security (NSS 2013)*, Madrid, June 2013.

[45] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.